Example of a Safety-Critical Element With Deliberately Unreliable Function

RVS White Paper 8

Peter Bernard Ladkin

20150101

An Example of Critical Software with Deliberately Subverted Critical Functionality

Rainer Faller recounted an example of a 32-bit processor, intended to be used for some critical operations, in which a "Debug/Maintenance" mode had been built in to the critical software (henceforth SW). In this mode, various safety functions were inoperational, and indeed in response to "normal" inputs the SW could engage in behavior which was dangerous on the system level. This D/M mode was not described in the SW documentation, and was reachable only through a specific 32-bit number given as input to the SW. Examples of benign such operations in non-critical contexts are often known as "Easter Eggs".

The question is whether a statistical approach to SW assessment, in which the SW is given as a "black box" and in which the SW is conceived as a Bernoulli process, can be used to assess such SW. For in order to conceive of the SW as a Bernoulli process, the SW must be "stateless" in operation, that is, its operation is dependent solely upon the value of the current input and not upon any internal SW state. Put loosely in more formal notation, if F(t) is the function describing the operation of the SW at time t, S(t) is the (partial, relevant) internal state of the SW at time t, and (t+1) is an incremental time step after time t, it is normally the case, for "stateful" SW, that F(t+1) = f(Input(t), S(t)), whereas for "stateless" SW, F(t+1) = f(Input(t)).

In the document <*Examples...., Ladkin-Littlewood 20141230*>, Bev Littlewood and I have pointed out that the statistical approach to SW qualification[1] based on the conception of SW operation as a Bernoulli process is not restricted to literally stateless SW. Many control system processes are cyclic, in that they take as input and retain values of critical parameters, take action based on that sequence of values, and regularly return to an initial state in which the sequence is no longer retained in the internal state. One common class of cyclic functions monitor the values of a critical parameter and trigger a safety function based on rapid changes in the value of that parameter. Such processes measure and act on, in effect, the value of the first derivative of the parameter, which is calculated discretely over a time period known in computer networking as a "sliding window". Such cyclic processes can also be interpreted as Bernoulli processes. One common example of such a process is a monitor which checks the internal temperature of a chemical reactor vessel and invokes a mitigation function if the temperature rises too quickly.

In other words, there are transformations which render stateless SW functionality that is prima facie stateful. These transformations are in first order mathematical, but the mathematics they implement is routinely implementable in SW. When such a transformation exists which renders stateless SW which is prima facie stateful, I call the SW "equivalently-stateless". Equivalently-stateless SW operation is a Bernoulli process and thereby subject to the usual statistical assessment, as given say in IEC 61508-7:2010 Annex D.

The SW with D/M mode is clearly not literally stateless. Its behavior at time (t+1) is dependent on whether the D/M bit sequence was input at time t. Let us for the purposes of argument assume that

- it is equivalently-stateless

1 "Qualification" is the term used here for the process by means of which SW is deemed acceptable for intended use.

- its operation in non-D/M mode is perfect according to its safety requirements specification
- its operation in D/M mode guarantees that a safety function will fail
- its operational profile is the uniform distribution of input
- its operational profile is so-called "continuous or high-demand mode" in IEC 61508-1:2010

What SIL Could Such Software Satisfy?

I ask what the statistical assessment of the SW can tell us about its suitability for operation from SIL 1 up to SIL 4 levels of reliability of safety-function execution.

First, note that operation in D/M mode entails the failure of a safety function. Let us further assume that there inevitably ensues a failure to satisfy the safety requirements specification (SRS). Note that this assumption is not redundant - it is not inevitable that the SRS will be violated even if a safety function fails.

Second, assume conservatively for the purposes of argument that the 32-bit sequence that triggers D/M mode is the only input (in most cases, input will be broader). Note that there are about 4×10^9 different 32-bit sequences. Assuming the uniform distribution of inputs[2], the likelihood that any one sequence will be chosen is about 2.5×10^{-10}.

Third, it follows according to the assumptions made above that the SW element[3] operates correctly according to the SRS, that is, it executes all its safety functions correctly, with a likelihood of $(1-(2.5 \times 10^{-10}))$ and fails to fulfil the SRS with a probability of (2.5×10^{-10}). The requirements of SIL 4 operational reliability for its safety functions specifies that they shall operate correctly with a likelihood of less than 10^{-8} . $10^{-8} > 2.5 \times 10^{-10}$. It follows that SIL 4 operational requirements are satisfied.

Note that there is a slight variance here with the literal requirement for SIL 4 operation as given in IEC 61508-1:2010 Table 3. A SIL 4 operational profile also requires that the safety function fails with a likelihood higher than 10^{-9}. I see no good reason why one would ever want an arbitrary safety function to fail. I therefore see no good reason why one would require a safety function to fail with any particular lower probability bound. I judge this Table 3 requirement to be anomalous and am ignoring it here (apart from in this paragraph).

Should Such Software Have Any Place in Safety-Critical Systems?

The question whether such a piece of SW (an "element" in IEC 61508:2010 terminology) should be operating in a safety-critical system is of course a different question from whether it fulfils SIL requirements in terms of statistics. For, despite the anomalous lower-probability-bound requirement for SIL 4, there is, or I would argue should be, a strong inclination not to use SW which is deliberately designed to have safety functions fail in certain circumstances. Rainer Faller has indicated that he has encountered such examples professionally, namely SW with D/M mode switches as described above. Many common operating systems are considered to have in them somewhere – see the 1988 example below. Yet many people also suggest the use of these OS's for critical systems, claiming they have been "verified through extensive use". This is a mistaken judgement in my opinion and in the opinion of others expert in the statistical assessment of software. What would be surprising is that somebody responsible for assessing a safety-critical system should think that such a design was appropriate for safety-function implementation.

2

3 "Element" is the IEC 61508 word for a piece of the system, an identifiable entity. It can be HW, SW, or a combination.

Indeed, there is a public example of such SW in a reliability-sensitive environment from 36 years ago. The so-called Morris worm brought down the nascent Internet in the US for a number of days in November 1998. One of the mechanisms through which the work propagated itself was through the debug-option-enabled version of the email transfer program Sendmail, designed and originally programmed by Eric Allman in Berkeley. Allmann had built a compile option in to Sendmail which enabled "debugging mode". When the Sendmail program was compiled with the debug option "set" (through a flag on the compilation shell command line), it enabled privileged access to the Sendmail program and its operational environment from a remote machine (that is, a different machine from that on which Sendmail was running). Such a compilation with debug "set" enabled the Sendmail developer inter alia to work on Sendmail from another location than hisher on-site office when convenient. Some commercial vendors of Unix systems had derived their versions from the BSD version of Unix developed at UC Berkeley and had distributed binary versions of Unix with their machines in which the Sendmail program had been compiled with the debug option "set". This of course was never the intention when the debug option was implemented – it was intended that all operational versions of Sendmail would not enable remote administrative access to the program source or operation. This was one of those occasions on which it became clearer that networked computer systems would rely more on Murphy's Law for their operation than they would on the good intentions, good will, understanding and mutual trust of their users and administrators. The Morris worm propagated itself in part through using the compiled-in Sendmail debug-option functionality in commercial Unix distributions.

I also note that the examples above are functionally identical to those of so-called "Easter Eggs", in which whimsical functionality is triggered through use of specific inputs. While D/M mode is "serious" while Easter Eggs are whimsical, the cases are functionally identical in terms of response to input and thereby in terms of functional reliability.

What IEC 61508:2010 Has And May Have To Say About Such Examples

It is not clear what IEC 61508:2010 has to say about safety functions with failure modes *deliberately* built in. By "failure" and "failure mode" I here mean: safety-function operation which fails to fulfil the SRS for the element; respectively a characterisation of the relevant internal and external environment accompanying the failure. I claim it would be an advantage if a future version of IEC 61508 provided clear guidance. I further claim that the guidance should be that such SW must explicitly include features such as D/M mode in its design specification and that these features must explicitly be taken into account during assessment as fit for purpose according to IEC 61508.

In a note Ladkin20100119 (*Comments on IEC 61508 Part 3 and Some Proposals for Modification*, internal document DKE 914.0.3_2010-0001, substantially similar to the publicly-available paper *Functional Safety of Software-Based Critical Systems*, http://www.rvs.uni-bielefeld.de/publications/Papers/LadkinAdaConnection2011.pdf , accompanying a Keynote talk to the 16th International Conference on Reliable Software Technologies, Edinburgh, June 2011), I pointed out that it was routinely possible

- to specify safety requirements formally, that is, to write a formal SRS,
- to check such a formal SRS for consistency and for some forms of relative completeness,
- and to check formally whether the design of an element as given in a design specification (SDS) fulfilled the formal SRS rigorously.
-

Such methods are indeed used routinely by industry attendees at the International Conference at which this paper was presented.

It is of course not possible that the SDS of such an "Easter Egg" design of safety functionality can be proven to fulfil the formal SRS. Indeed, part of the hypothesis of the example entails that the SDS violates the SRS.

A current proposal from myself and Bernd Sieker under discussion in the Safe Software working group of the German electrotechnical standards organisation DKE (DKE AK 914.0.3 "Sichere Software") is that

- there are 26 different properties of SW and its documentation that may be guaranteed by current industrially-practical formal methods (call these the "26 methods"),
- the 26 methods be explicitly enumerated as industrially-applicable formal methods in the next version of IEC 61508 (the maintenance period is predicted to start at the end of 2015, so the version could be expected in 2018 – I refer to it forthwith as IEC 61508:2018projected),
- the current designation in IEC 61508-3:2010 of generic "formal methods" as in general "Highly Recommended" (HR) for SIL 4 should be elicited specifically for each of the 26 methods
- use of the 26 methods should be HR for all SILs

The occurrences and current recommendations concerning what are often referred to as "formal methods" are in IEC 61508-3:2010 Annex A, namely

- Table A1 line 1b, "Formal methods";
- Table A2 line 11c, "Formal design and refinement methods";
- Table A4 line 1c, "Formal design and refinement methods";
- Table A5 line 10, "Formal verification";
- Table A9 lines 1, "Formal proof" and 3, "Static analysis"

These terms are all otherwise undefined in IEC 61508:2010. Yet they beg for clarification. What, for example, is the difference between "Formal verification" and "Formal proof"? (If there is no difference, why isn't the same term used?) A proof has an object. It is a proof *of some assertion* and which assertion this is is crucial to the efficacy of a proof. If I want to prove that 2+2=4, then a proof of 0+0=0 will not suffice. So "Formal proof" may be HR for SIL 4, but what exactly is it that should be proved? I may note that it is unfortunately becoming common that some software is said to have been "formally proved", but such an assertion does not necessarily mean that said software has been exceptionlessly demonstrated to fulfil its requirements specification. But suppose it has. Is it that the source code, under a given semantics, has been so shown to fulfil its requirements specification, or is it that the compiled and linked object code has been so shown? If the former, it does not follow that the compiler and linked object code fulfils the requirements specification unless there are additional guarantees on the behavior of the compiler and linker with respect to the given semantics of the source code.

Use of the terminology of the 26 methods will alleviate such unhelpful ambiguity. Use of the 26 methods themselves will give suitable assurance that the SW operates to its intended purpose. Any SIL level is critical to some specified extent; I see no justification for an HR qualification for a SIL 4 application that would not also apply to a SIL 1 application of the same element.

Conclusions

It is difficult to see how IEC 61508:2010 applies to such "Easter Egg" elements. Although a 32-bit example under the uniform distribution of inputs fulfils the reliability requirements for SIL 4 safety functions, it is clearly more desirable that a trigger of D/M mode is rendered impossible during normal operation. An application of the 26 methods would render proposed safety-related software contains such undocumented features unable to pass the assessment processes for fitness for purpose, assuming that the assessors agree with the second sentence of these conclusions. It follows that highly recommending the use of the 26 methods would eliminate the uncertainty in a literal reading of IEC 61508:2010 as to whether such Easter Egg elements are appropriate to implement safety functions.

Acknowledgements

I thank Rainer Faller for perceptive comments on an earlier version, and for his agreement for this discussion to be published as a White Paper. I thank members of my RVS group a tthe University of Bielefeld for helpful discussion of the example.