Causal Analysis of the 1991 Patriot Missile System Failure

Peter Bernard Ladkin
20150511

The Incident

The US General Accounting Office reported to the Chair of the House Committee on Oversight and Investigations in February 1992 on the failure of a Patriot anti-missile system to engage and destroy an incoming hostile Scud missile at a base in Saudi Arabia in February 1991. The report may be found at http://fas.org/spp/starwars/gao/im92026.htm . The incident is described in the report's first sentence as follows

> *On February 25, 1991, a Patriot missile defense system operating at Dhahran, Saudi Arabia, during Operation Desert Storm failed to track and intercept an incoming Scud. This Scud subsequently hit an Army barracks, killing 28 Americans.*

There is a slide set on the incident available from Matthew Squair at
https://msquair.files.wordpress.com/2015/05/patriot-timing-error.pdf

How the Incident Occurred

The results of the inquiry, in brief, are as follows

> *The Patriot battery at Dhahran failed to track and intercept the Scud missile because of a software problem in the system's weapons control computer. This problem led to an inaccurate tracking calculation that became worse the longer the system operated. At the time of the incident, the battery had been operating continuously for over 100 hours. By then, the inaccuracy was serious enough to cause the system to look in the wrong place for the incoming Scud.*
>
> *The Patriot had never before been used to defend against Scud missiles nor was it expected to operate continuously for long periods of time. Two weeks before the incident, Army officials received Israeli data indicating some loss in accuracy after the system had been running for 8 consecutive hours. Consequently, Army officials modified the software to improve the system's accuracy. However, the modified software did not reach Dhahran until February 26, 1991--the day after the Scud incident.*

The Function of the Patriot Missile System

The function of the system can be described in general terms as follows.

1. Detect and "acquire" an incoming target (hostile missile);
2. Track the target until it is within "shootdown" range (a series of parameters);
3. Fire a counter-missile;
4. Guide the counter-missile to its target;
5. Explode the counter-missile within a range to destroy the target.

Where the Failure Occurred

The “loss in accuracy” concerned the tracking of an incoming target. It is Steps 2 and 4 in which the failure occurred. An area called a “range gate” is determined around the detected target by the system, and the success of Step 4 depends upon the target being acquired and “held” (continuously tracked) within the center of the range gate. When the target is near the periphery, or outside, the range gate at the time of firing (Step 3) then Step 4 is less probable and Step 5 is not necessarily accomplished, which is what happened in the 1991 incident.

Why the Failure Occurred

The report explains the errors in calculation of the target’s position as follows:

> *The range gate's prediction of where the Scud will next appear is a function of the Scud's know velocity and the time of the last radar detection. Velocity is a real number that can be expressed as a whole number and a decimal (e.g., 3750.2563...miles per hour). Time is kept continuously by the system's internal clock in tenths of seconds but is expressed as an integer or whole number (e.g., 32, 33, 34...). The longer the system has been running, the larger the number representing time. To predict where the Scud will next appear, both time and velocity must be expressed as real numbers. Because of the way the Patriot computer performs its calculations and the fact that its registers are only 24 bits long, the conversion of time from an integer to a real number cannot be any more precise than 24 bits. This conversion results in a loss of precision causing a less accurate time calculation. The effect of this inaccuracy on the range gate's calculation is directly proportional to the target's velocity and the length of time the system has been running. Consequently, performing the conversion after the Patriot has been running continuously for extended periods causes the range gate to shift away from the center of the target, making it less likely that the target, in this case a Scud, will be successfully intercepted.*

Questions Arising from Informal Analysis

There was some discussion of this on the System Safety List. Martyn Thomas asked
http://www.systemsafetylist.org/1745.htm

> *Why did they need to keep the time, over a period of 100 hours, in order to determine how far away an incoming missile is? The two things appear to me to be unrelated.*

Martyn elaborated privately

> *The RGA [range gate] is a function of time, but [it is not clear] why "time" has to be the elapsed time since the system was started, rather than (say) gps time, or time since the start of the current radar scan, or ...*

I propose that there is indeed an anomalous design feature; indeed two. The first is that there are errors in a parameter which is used for targeting, Steps 2 and 4, which arise from the fact that the required data is poorly represented and calculated. Modern software engineers would call it an inappropriate data-type coercion: a data type was represented in a format which didn’t allow for appropriate calculations with that type. The second is that the parameter which was chosen, clock time since system boot-up, is not integral to the physical calculation required, as Thomas clearly points out, mentioning also some obvious alternatives.

There are other questionable phenomena which arise. The report suggests that the system was not intended to be "up" for so long – it needed to run virtually continuously in this deployment. However, it is not said whether there was a precise time frame for system function specified in the system requirements. The impression from reading the report is that there was no informal "intention" for the system to be operating so long that the data-type errors accumulated to hinder the functionality, but it is not said that a limited time frame was specified in the system requirements.

More Formal Causal Analysis

The key event was that Step 5 was not fulfilled. The Patriot system did not bring down the Scud which had been acquired. And the reason for that is that the counter-missile was mis-aimed: Steps 2 and 4 were faulty. And the causal reason for the faultiness of Steps 2 and 4 was the "clock drift", more precisely the accumulated calculation error due to the inappropriate data-type coercion.

Let us denote these more formally:

- KeyEvent: Step 5 failed: the counter-missile failed to destroy the Scud
- SubKeyevent: Steps 2 and 4 failed: the counter-missile was mis-aimed
- TypeFault: calculations performed on the computational data type did not represent the physical reality, due to inappropriate type coercion.
- InappTypeCoercion(Parameter): the parameter was represented by a data type, the results of whose operations were not homomorphic with the physical operations on the parameter;
- MisCalc: the software miscalculated the values of the physical parameter about which the software was calculating
- ElapsedTime: Elapsed time since system start

Let me also define some potentially useful alternative phenomena:

- RelativeTime: Any version of time appropriate for the tracking task (c.f., the varieties suggested by Thomas)
- TypeCorrect(Parameter): a data type was used to represent Parameter, the results of whose operations were homomorphic with the physical operations on Parameter

WBA uses the notion of Necessary Causal Factor (NCF), and phenomenon A is said to be a NCF of phenomenon B, in symbols A []-> B, if A satisfies the Counterfactual Test (CT) with respect to B. A satisfies the CT with respect to B if A and B both happened, but in the nearest "possible worlds" in which A didn't happen, B wouldn't have happened either.

- A []-> B iff A satisfies CT wrt B;
- A satisfies CT wrt B iff in the nearest possible worlds in which A did not happen, B did not happen either;
- A NOT-[]-> B means (is shorthand for) NOT(A []-> B).

The terminology of nearest "possible world" comes from modal logic, but what it concretely means is the way the world would have been had A not happened, but everything else had stayed as closely similar to the way things happened as possible. For example, it is a "near possible world" in which I might have been named "Frederick". That was a name of my father, and my parents just had to have chosen that instead of "Peter", maybe on a whim. But a world in which I was named "Smith" instead of "Ladkin" is not as near – much more would have had to be different, for example my family would have had to have been named "Smith", and that would in turn have its causes in history.

A number of causal-factor assertions are immediately available:

- SubKeyEvent []-> KeyEvent
- MisCalc []-> SubKeyEvent
- InappTypeCoercion(ElapsedTime) & Use of ElapsedTime []-> MisCalc

We can also make some assertions about what are not, or would not be, causal factors.

- InappTypeCoercion(RelativeTime) & Use of RelativeTime NOT-[]-> MisCalc
- TypeCorrect(ElapsedTime) & Use of ElapsedTime NOT-[]-> MisCalc

But here the semantics of "[]->" is not quite as originally given. Use of the CT requires that both antecedent and consequent actually occurred. Neither (InappTypeCoercion(RelativeTime) & Use of RelativeTime) nor (TypeCorrect(ElapsedTime) & Use of ElapsedTime) occurred, because in the former case Use of Relative Time did not occur, and in the latter case TypeCorrect(ElapsedTime) did not occur.

We can formally accommodate this as follows.

It may be argued (and I would so argue) that appropriately professional software engineering ensures TypeCorrect(Parameter) for each Parameter used in a program. So TypeCorrect(ElapsedTime) is a phenomenon which should have occurred – there is a norm of professional software engineering saying it should have occurred – but which did not. So TypeCorrect(ElapsedTime) is a reified non-event in WBA. We can consider it as ontologically existing in the actual world and thereby may partake in the CT just as other actually-occurred phenomena.

Can (Use of Relative Time) equally so be considered a reified non-event in the actual world? I believe that Thomas would argue that yes, it can, on a basis similar to the following:

- appropriately professional software engineering uses quantities appropriate for the task at hand;
- Use of ElapsedTime is inappropriate for the task at hand (Steps 2 and 4);
- Use of RelativeTime is appropriate for the task at hand.

Priority of Non-Events

We have identified two non-events, Use of RelativeTime and TypeCorrect(ElapsedTime). There is also a third non-occurring event, namely TypeCorrect(RelativeTime). This is non-occurring for the same reason that InappTypeCoercion(RelativeTime) did not occur, namely that the parameter RelativeTime was not used in the program.

WBA itself is not concerned with assigning priority, relative importance, to NCFs of KeyEvent. We might consider some formal criteria by means of which such relative importance can be assigned. However, it appears that nothing can be obtained from the purely formal consideration of causality by itself. Observe that the two statements of non-NCF above (designated with NOT-[]->). If you use some version of relative time, you are OK even with the inappropriate data typing, and if you use appropriate data typing, you are OK even with inappropriate choice of time parameter. Pure causality gives no formal "hook" on which to hang a distinction which might lead to a priority assignment to the type of fault. Such a priority assignment must come therefore from the subject matter, namely Software Engineering.