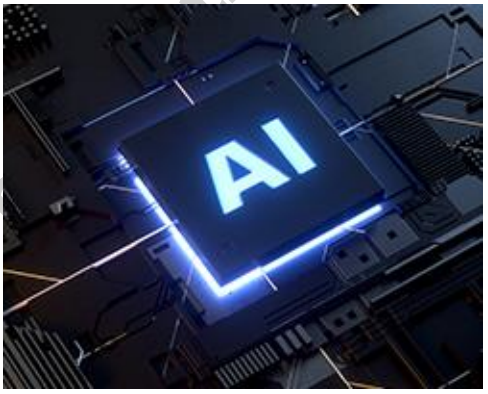
www.pipelinepub.com
Volume 20, Issue 12

Network Reliability and GenAI

By: Mark Cummings, Ph.D.



The CrowdStrike BSOD (Blue Screen of Death — Windows crash and failed reboot) incident is a warning of things to come. It is a symptom of how we rely heavily on networks that are increasing in scale, complexity, and volatility. It is only going to get worse with wide scale implementations of GenAI. To effectively respond, innovative ways to monitor, prevent, and fix problems will need to be developed and deployed.

The CrowdStrike Incident in Perspective

The old Telco wired networks were designed to a "5 nines" reliability standard, meaning they would run reliably 99.999 percent of the time. To state it another way, in 1 million minutes (roughly 694 days) of operation, the network would not have more than one minute of downtime. In the CrowdStrike incident, downtime ranged from hours to days, and in the case of Delta Airlines over a week. If the incident were the only CrowdStrike outage, and CrowdStrike met the 5 nines reliability standard, it would have had to be in continuous operation for at least ~1 million days (~42,000 years).

CrowdStrike didn't create a problem in its own product. Rather it created a problem in other products. It can actually be argued that the CrowdStrike product worked fine throughout the whole incident.

The effects of the problem were widespread and concerning. In addition to disruption of airlines, emergency services such as 911 in the U.S. and similar services around to world went down. Hospitals and emergency rooms had trouble providing services. Delivery services were impacted. Multi-mode freight transportation systems were affected. Effects rippled across networks, even in those that were not running affected computers. We were lucky it happened leading into a weekend in summer vacation season. If it had started at the beginning of a week in a busy time of year, things could have been much worse.

The big problem is that CrowdStrike is not the only infrastructure software product in our networks. A friend of mine did an analysis of a typical online banking transaction and found that there were 176 software products involved in completing a transaction. How many software products are involved when pumping a gallon of gas into your car? In having water come out of your tap when you turn on the faucet? Your toilet empty when you flush? Or a loaf of bread being on a store shelf when you reach for it?

Our quality of life is built on this growing number of very large scale, very complex, and very volatile systems. A problem with any one of the many thousands of software elements under the control of a profusion of different companies can do a lot more than what happened in the CrowdStrike incident. This is why the CrowdStrike incident is a concerning harbinger of things to come.

There also are clear cybersecurity issues involved here. Here, however, we will focus on understanding what the problem was in the CrowdStrike incident, how GenAI may exacerbate these types of problems, and what can be done to avoid more of them in the future.

The CrowdStrike Problem

CrowdStrike provides a security product that runs on Windows computers. The product is a system based on patterns (called signatures) of previous attacks. That is, attacks that have occurred, been detected, and for which signatures have been developed. There are so many kinds of these often called zero-day attacks that CrowdStrike has to send out signature updates every two hours. This fact alone indicates the volatility of today's networks. Given that new attack types can proliferate across very many targets in seconds, and do significant damage in minutes, my guess is that customers of CrowdStrike would like updates even more frequently.

CrowdStrike checks the content of the updates to make sure they are okay. Because of the way Microsoft Windows is set up, these updates have to go into the kernel (foundation of the OS). What happened in this incident was that the update file had an incorrect file name extension. Checking the contents didn't, and never would have, detected the error. When the message was received, it was interpreted (because of file extension name) as a code update not signature data. Loading it into the kernel as code caused the OS to crash in a manner that was not easily fixed. So, the computers kept trying to reboot and kept failing. It took a lot of manual effort (often with privileged physical access to the machines) over a significant amount of time to get all the computers involved up and running.

We may never know for sure why there was an error in the file extension name. It could have been a human error (fat finger problem), a system error in the chain from file creation to file transmission, a system error in file preparation, or something else. What is clear is that it was an unexpected error type. Most testing tools look for types of errors that have been seen before (known unknowns). Unexpected error types are not caught. The CrowdStrike error was unexpected, therefore nobody had put in control measures for the possibility of its occurring.

What is the probability of an unexpected type of error occurring? We know from the CrowdStrike incident that it is not zero. The more scale, complexity, and volatility we have in our networks, the higher the probability of encountering unexpected error types.

GenAI and Future Problems

We are now introducing GenAI into this environment of scale, complexity, and volatility. Introducing anything fundamentally new to such an environment increases the probability of an unexpected error type being encountered. The introduction of GenAI, by its nature and in its current state of evolution, increases the probability of unexpected errors even more. This is because GenAI is being used in software development, parameter setting, and other areas that affect the operation of our networks.

In software development, GenAI has three characteristics that, in combination, make it much more likely to produce unexpected error types. GenAI will:

- Make errors that are very different from the type that human coders make.
- Hallucinate in the software development process.
- Make hallucinations in software development appear fine.

Experience with previous forms of automation indicate GenAI will produce many previously unknown problems due errors, or types of errors, that human software developers have not previously made or are not likely to make. The hallucination problem is well known but more problematically subtle. As a result of that and other quality concerns, everyone agrees that GenAI-developed software must be reviewed by people. But as time pressures, technical pressures coming from increased complexity, and attempts to control costs mount, the temptation to trust GenAI is going to be hard to resist. And, on cursory inspection, it always looks good. That's the problem.

The same factors are at work in parameter setting and other kinds of network operations. For example, before GenAI, we saw an incident in which a fat finger problem in manual operations brought down Google's S1 network in the Northeastern US. This shows that parameter and operational errors can cause a crash. We can foresee other opportunities for errors with the introduction of GenAI into the mix.

Many software updates and parameter setting updates occur in our networks each day. Even a very small percentage (back to five 9's again) of unknown problems can cause widespread crashes. Some of those crashes will be merely inconvenient. A few can be as bad as CrowdStrike or worse.

Guarding Against Future Problems

Our systems and networks were already struggling with scale, complexity, and volatility. With the addition of GenAI it is only going to get worse, even as we are becoming more and more dependent on our systems and networks. The way out is to develop new ways of making sure that what we think are "small changes" don't result in crashing everything down.

We have solutions for the known types of errors. They're not all perfect, but they're a good start. Another approach, called "sandboxing" has been around for a long time, but is now beginning to be more recognized. In this approach, a small network/computing environment similar to the operational environment is created in a separate domain. A domain where undesired effects cannot pass through into the operational environment. Into this sandbox the new software, or parameter setting, is inserted and the results are observed. Special attention is paid to any unexpected outcome.

Sandboxing as it is practiced today can catch some of the unexpected previously unseen problems. Simple sandboxing might have caught the CrowdStrike problem. So, the first recommendation is that vendors and large user organizations should develop sandboxes. Vendors should put things they are planning to ship into their sandbox before shipping. Large organizations should put incoming changes into their sandbox before installing. These measures will help a lot.

The problem we face today is that most sandboxes don't accurately reflect the current state of the network. They tend to be over simplified, small, not rapidly changing abstractions of the operational network. It is hard and expensive today to create sandboxes that reflect the full scale, complexity, and volatility of our networks. This is where innovation must come in. We need to develop ways of making our sandboxes accurately reflect the up-to-the-minute (second?) representation of our networks in all their complexity and scale. And we need to find a way of doing this economically. The alternative is more crashes as bad or worse than the CrowdStrike incident.

Conclusion

If we want to preclude events like CrowdStrike BSOD incident from happening in the future we have to find innovative ways to monitor, prevent, and fix problems. Sandboxing may be the best way to do this. But innovation is needed to make sure the simulated sandbox network effectively simulates the operational network. More specifically, that it is representative of the operational network's: *Volatility*,

by being very frequently updated; *Complexity*, by having representation of the full range of systems; and *Scale*, by developing innovative ways using statistical methods to effectively duplicate the size of large networks. Vendors and large end user organizations forming partnerships with innovators is the best way to get to the environment we want and need.