For Good Measure
Curves of Error

DAN GEER

Dan Geer is the CISO for In-Q-Tel and a security researcher with a quantitative bent. He has a long history with the USENIX Association, including officer positions, program committees, etc. dan@geer.org

> The major difference between a thing that might go wron-g and a thing that cannot possibly go wrong is that when a thing that cannot possibly go wrong goes wrong it usually turns out to be impossible to get at or repair. —*Douglas Adams*

One hears often enough that the error rate for software is so many flaws per thousand lines of code or the like. A fraction of those flaws turn out to create vulnerabilities. A fraction of those vulnerabilities get exploited. And "we" learn about a fraction of those exploits. Let's call it

$$S * F * V * E * P$$

In other words, we create S lines of new code, F of which are wrong, V of which are vulnerabilities, E of which are weaponized, and P of which come to our attention. Let's stipulate one thing: arguing about what constitutes a line of code is irrelevant. While we're at it, let's stipulate that everything here is subject to argument about definitions and what goes in what set.

That kind of formulation is similar to the kinds of rough calculations around whether there is other intelligent life in the galaxy. On the one hand, there are something like 100 billion stars in the Milky Way. On the other hand, intelligent life requires a bunch of pretty unlikely coincidences (probabilities) multiplied by that 100 billion. You only need five 1% probabilities multiplied together to get down to as many intelligent life planets in the Milky Way as you have fingers. Six such conjunctions and the odds turn against our very existence.

So, how many lines of code? That is harder to estimate than the number of stars in the Milky Way. An unsubstantiated claim in CSO magazine [1] was that 111 billion lines of code (LOC) were created in 2017. Elsewhere, there's a slightly more substantiated estimate of 20 million developers at work today [2]. The old rule of thumb for a developer is/was 50 LOC/day or 10–15 KLOC/year. Multiplied times 20 million devs, that's 20–30% of that 100+ billion LOC claim. Of course, some code is not written by hand but by machine, but is it really 70–80%? Or are there more than 20 million devs? Or are they more productive than 10–15 KLOC/year? With almost 8 billion people on earth, does it sound right that 1 in 400 is a dev? Let's take S = 10^{11} for the moment.

How many flaws are in that code? The old rule of thumb is/was 25–50 flaws per thousand lines of code (KLOC), although the various measurements that come to that range of numbers were for software products that are deployed en masse after a defined build process as opposed to continuously thrashed web applications. Anyhow, if we use 40 flaws/KLOC and multiply it by 100 billion LOC, we are looking at 4 billion new flaws per year.

As a kind of comparative calibration, the top six open source package repositories account for 80% of all open source repositories [3], a combined total of 1.75 million packages, which number is increasing by 1,000 per day. That's 23% annual growth, and we're not even talking about GitHub or SourceForge.

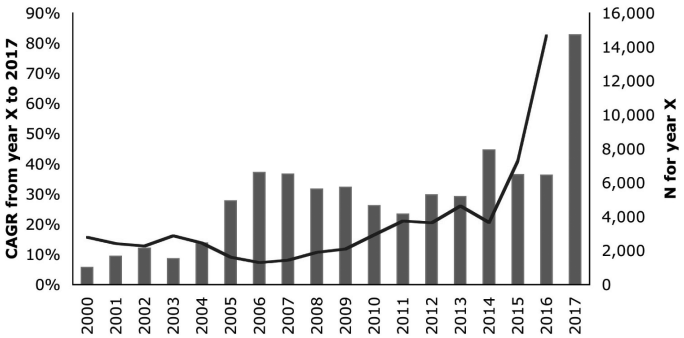

Figure 1: Number of CVE entries and CAGR from each year to the 2017 value

Turning to what fraction of uncategorized flaws are security flaws, i.e., what fraction of bugs create vulnerabilities, we find two schools of thought. School One: any and all bugs are vulnerabilities unless and until proven otherwise. School Two: only a small fraction of all bugs are security bugs. For today's purpose, we'll side with School Two, taking the line that vulnerabilities are a small percentage of total flaws. That may well be incorrect in the sense of "failing to make the conservative assumption" (conservative with respect to security outcomes, that is).

In any case, the vulnerability to flaw ratio is related to Bruce Schneier's foundational question of whether, in truth, vulnerabilities in software are sparse or dense [4]. Chris Wysopal says that Veracode [5] finds 0.1 vulnerabilities per KLOC, so a 40 flaws/KLOC starting point means that 0.1/40 = 0. 0025 or 1/4 of 1% of flaws are actually vulnerabilities. That makes it conservative to say that 1% of all software bugs are vulnerabilities, so we'll go with that for the moment.

Then there is the probability that a given vulnerability can and will be weaponized, which is to say turned into a deployed exploit. As Dave Aitel [6] has repeatedly argued, what is sparse is not vulnerabilities that could be weaponized but the people who can weaponize vulnerabilities. Add to that that good exploits may require more than one vulnerability, i.e., the conversion rate of vulnerabilities to exploits may be lower still. Plus in a whole-world setting where the installed base of software is growing faster than the human population, then the fraction of vulns that are weaponized might actually be falling, not for want of opportunity but for want of labor (back to Aitel). In any case, and for the purpose of argument here, let's call it 1-in-200 or .005 that a given vuln will be weaponized.

That 1-in-200 is almost surely way conservative. Brian Martin of Risk Based Security [7] has data showing that out of 199,311 vulns with a CVSSv2 9.3 or higher, 6,244 have a public exploit, 2,350 have a proof of concept, and 3,048 have a private exploit. That works out to 5.8% of those (very serious) vulns are known to have, or be capable of, exploits. That's an order of magnitude higher than 1-in-200, but we'll stick with 1-in-200 for the moment. While we're at it, HackerOne says that in 2018 they managed 78,275 reports [8]. If even half of those are valid security bugs, then it would more than double the 2018 CVE or VulnDB count.

Of course, some weaponized vulns will never come to our attention. For this estimate, we must admit that we are in the murk; zero-days don't get counted since they aren't 0days if we can count them, nor are exploits that are use-once for precious targets something we'll ever see. This is what Ablon and Bogart covered so well for RAND [9]. For the sake of argument, let's pick maximum ignorance priors, i.e., say that 50% of vulnerability weaponization is unobservable while 50% comes to some kind of public attention. This brings us back to the top, viz.,

$$S * F * V * E * P$$

which we'll rewrite with $S = 10^{11}$, F = .04, V = .01, E = .005, and P = .50

$$10^{11} * .04 * .01 * .005 * .50 = 100{,}000/\text{yr}$$

A number like 100,000 de novo, non-targeted exploits in the wild per year is certainly a stunning number. But is it real? Does it carry policy freight?

In calendar 2017, there were 14,714 new CVE reports made. Going back to year 2000, there were 1,020. That works out to a compound annual growth rate (CAGR) from 2000 to 2017 of 15.7%. For 2001, there were 1,677 reports, which amounts to a CAGR from 2001 to 2017 of 13.6%. Figure 1 lays this all out; for each year, the column represents the number of CVE reports made, and the line is the value for the compound annual growth rate between that year to 2017.

Focusing on the longest term, from 2000 to 2017, that CAGR of 15.7% begs the question: how fast is the total body of installed code growing? If that total installed base is growing faster than 15.7%, then CVE would say that either we're collectively getting better at making new software secure or we're collectively getting worse at finding (and recording) security problems in new software.

Generally speaking, a measure that has constant error will return value estimates that are wrong, but its trend line will at least have the right shape. Therefore, we might ask the question, "Does CVE has relatively constant error?" Unfortunately, the answer to that question is almost surely "No"—CVE's coverage has been shown to be poor, and the recent spike in its listing of new vulnerabilities is more a response to embarrassing congressional hearings in late 2016 than anything else: i.e., the big jump in 2017 is an artefact.

We seem to have no good measure, then, of the shape of the curve of error. Bug bounty payouts are not it (it being the way to measure the fraction of vulnerabilities that are exploitable). So we are back to the number to noodle over: is finding 100,000 de novo, non-targeted exploits in the wild per year a round-number estimate good enough to inform policy?

Obviously, spreading 100,000 new exploits through 100 billion new lines of code constitutes a seemingly low density—literally one in a million ($10^5/10^{11}$). That ought to be reassuring. Or should it? The 2016 Ford F150 pickup truck is said to have 150 million lines of code [10]. By our working guess of 1-in-a-million, that $150 * 10^6$ LOC might be expected to set the stage for 150 exploits. Perhaps thankfully, the Boeing 787 Dreamliner is said to have 7 million LOC which gives the naive estimate of a half-dozen exploits waiting to come "out of the nowhere into the here."

If the reader thinks software errors are some flavor of inevitable and are not designed in by a present-day Illuminati, then those errors are sprinkled over software products like some kind of pixie dust. Even if the estimates above are off by an order of magnitude (in either direction), the implication, both personal and policy, might well be this: the more software there is in a product, the less you should depend upon it. The more a given supplier lards up the product with features, the less you should want to depend on it. The more often the software base turns over, the less the software in it has been burned in.

For the present author, a state of security is the absence of unmitigatable surprise, hence the Douglas Adams quote at the start, the conservative assumptions throughout, and the previous paragraph's appeal to retaining alternative—analog if you prefer—mechanisms. That position, itself, would be shown to be conservative if finding bugs with AI turns out to be as effective as it might be. While AI uber-bug-finding would likely depress the number of latent, as yet invisible threats, it is hard to imagine that any substantial entity would be prepared for their AI to autonomously fix bugs it had autonomously found, so, in turn, the number of known bugs could well increase faster than the market cycle could accommodate fixing them. And then we'd be writing about this in parallel to the vaccination of school-age children as a state-imposed access requirement to a public good, only now it would be autonomous update as a state-imposed access requirement to a different public good.

References

[1] S. Morgan, "World Will Need to Secure 111 Billions Lines of New Software Code in 2017": https://www.csoonline.com/article/3151003/world-will-need-to-secure-111-billion-lines-of-new-software-code-in-2017.html.

[2] R. Cox, "How Many Go Developers Are There?": https://research.swtch.com/gophercount.

[3] Open source repository timeline: http://www.modulecounts.com/modulecounts.csv.

[4] B. Schneier, "Should U.S. Hackers Fix Cybersecurity Holes or Exploit Them?" May 19, 2014: https://www.theatlantic.com/technology/archive/2014/05/should-hackers-fix-cybersecurity-holes-or-exploit-them/371197/.

[5] Veracode, *State of Software Security*, vol. 9: https://www.veracode.com/state-of-software-security-report.

[6] D. Aitel, personal communication.

[7] B. Martin, personal communication; https://vulndb.cyberriskanalytics.com.

[8] "Hacker-Powered Security Report 2018," July 11, 2018: https://www.hackerone.com/blog/Hacker-Powered-Security-Report-2018.

[9] L. Ablon and A. Bogart, *Zero Days, Thousands of Nights: The Life and Times of Zero-Day Vulnerabilities and Their Exploits*, RAND Corporation, 2017: https://www.rand.org/pubs/research_reports/RR1751.html. Also available in print form.

[10] R. Saracco, "Guess What Requires 150 Million Lines of Code...," January 13, 2016: https://www.eitdigital.eu/news-events/blog/article/guess-what-requires-150-million-lines-of-code/.