For Good Measure

The Undiscovered

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

Do not become the slave of your model.
—*Vincent Van Gogh*

Précis: Security metricians must steal all the techniques we can—we don't have the time to invent everything we need from scratch. This column does just that, motivated by the question of whether patching matters, a question that just will not go away.

A common problem in wildlife biology is simply this: "How many X are there in Y?" as in "How many frogs are there in the pond?" The most common method is "capture-recapture." The technique is simple and has been long applied not just to biology but also to things as disparate as how to adjust the US Census for undercount [1] to how many streetwalkers there are in Glasgow [2]. As with any statistical technique, there are assumptions, which we come to in a moment. First, this is the core process for estimating N, the number of frogs in the pond:

1. Take a sample of size n_1 and mark them.
2. Release the sample back into the wild.
3. Wait a short time, a week perhaps.
4. Take another sample of size n_2, counting the m_2 of the second sample that are found to be marked.
5. As $\frac{m_2}{n_2}$ should be the same as $\frac{n_1}{N}$, conclude $N = \frac{n_1 n_2}{m_2}$

That formulation is called the "Lincoln Index." As an example: catch 41 frogs and band them, then, a week later, catch 62 frogs and find that six are banded; we therefore estimate that there are

$$\frac{41 \times 62}{6} = 424$$

frogs in the pond. (Note: it is better to actually calculate $((n_1 + 1)(n_2 + 1)/(m_2 + 1)) - 1$ which yields an estimate of 377 frogs in the pond.)

The assumptions behind capture-recapture studies are that catching and marking the frogs does not change their behavior, that marked frogs completely mix into the pond's population, that any one frog, marked or not, is equally likely to be caught, that sampling is quick (preferably all at once), and that the population did not change between captures.

A second method, called "removal-capture," follows this process:

1. Catch n_1 frogs during a fixed-duration hunt and remove them.
2. Wait a short time, a week perhaps.
3. Catch n_2 frogs during a second fixed-duration hunt.

4. Calling N_0 the number of frogs on day 0, if

$$\frac{n_1}{N_0} = \frac{n_2}{N_0 - n_1} \text{ then } N_0 = \frac{{n_1}^2}{n_1 - n_2}$$

As an example: the first catch finds 78 frogs and the second 57; we therefore estimate that there were

$$\frac{78^2}{78 - 57} = 289$$

frogs in the pond on day 0.

The assumptions behind removal-capture studies are that the population is reasonably static, large enough for a significant catch in each subsequent sample yet small enough that a reduction in catch will be noticed, and that within a constant time interval a constant fraction of the frogs will be caught.

So why am I mentioning all this? In a May 2014 article in *The Atlantic* [3], Bruce Schneier asked a cogent, first-principles question: "Are vulnerabilities in software dense or sparse?" If they are sparse, then every vulnerability you find and fix meaningfully lowers the number of vulnerabilities that are extant. If they are dense, then finding and fixing one more is essentially irrelevant to security and a waste of the resources spent finding it. Six-take-away-one is a 15% improvement. Six-thousand-take-away-one has no detectable value. Eric Rescorla asked a similar question in 2004: "Is finding security holes a good idea?" [4] Rescorla established that it is a non-trivial question, as perhaps confirmed by our still not having The Answer.

In other words, we want to know how many frogs (vulnerabilities) there are in some pond (a software product). One might then ask whether either or both of the capture-recapture and removal-capture techniques might help us answer Schneier's and Rescorla's challenge, the challenge of picking a policy direction based on whether vulnerabilities are sparse or dense. Do we want a policy that skips patching in favor of rolling software fast enough to make it a moving target? [5] If we decide to keep patching, are we better off disclosing or keeping the repairs secret?

Starting at what may be the beginning, in a 1993 paper Vander Wiel & Votta [6] gave capture-recapture for software engineering a good airing. Their body of study was on latent errors of design in software projects and whether multiple, parallel design reviews might be structured so as not only to find design flaws but to also estimate how many further design flaws were as yet undiscovered. In other words, their frog is a design flaw and their pond is the design of a software project. The context of their work was an attempt to improve on what had been a quota system for design reviews at Bell Labs—a design reviewer had to find between a fixed minimum and a fixed maximum number of faults per page of the design document.

The Vander Wiel & Votta paper is worth a read if you want early statistical details. Their basic result was to assess how violating the assumptions (that are appropriate for wildlife biology) affected using the capture-recapture technique to estimate the number of design flaws in a software project. Quoting from their paper:

> Our approach treats the faults found by reviewers preparing for a design review as data from a capture-recapture sampling scheme. We use a Monte Carlo simulation to investigate the inaccuracies of the capture-recapture estimators due to assumption violations when faults have varying detection probabilities and reviewers have different capture probabilities. Although we would like to use data from real world design reviews to perform this study, it is impossible. We can not control the fault detection and reviewer capture probabilities in design reviews, nor could we ever hope to obtain the number of reviews required to perform a statistically significant study.

The problem at hand for security metricians is parallel—we cannot do controlled experiments, our vulnerability finders have broad ranges of skills (plus the most skilled ones are not talking), and vulnerabilities range from trivial to find to the kind of impossible that wins the "Underhanded C" [7] contest.

Their simulations enabled Vander Wiel & Votta to make some general recommendations for mitigating violations of the statistical assumptions. One is for when faults are not equally easy to find—we have that problem with respect to vulnerabilities, and they tell us that if we can group faults such that those within a group *are* equally easy to find, then we can do capture-recapture for individual groups of faults so long as the groups are large enough "that some faults in each group are discovered by more than 1 reviewer." Another is for when fault finders are not equally skilled. We have that problem, too, and they tell us that grouping fault finders by skill level might be worthy of study. (They did not pursue that option, but perhaps "we" should.)

The two decades since 1993 have seen a lot of experimentation with capture-recapture in the software engineering literature. Petersson et al. [8] reviewed that history, classifying the assumptions (and their violation) as:

> M_0 the probability of a fault being found is the same for all faults as is the ability of each inspector to find each fault
>
> M_h the probability of a fault being found is the same for all faults, but detection ability can vary from inspector to inspector
>
> M_t the probability of faults being found varies, but inspectors all have the same ability to find each fault

M_{th} the probability of faults being found can vary and so can the ability to find a fault can vary from inspector to inspector

and, yes, each of these four needs a different modeling regime.

In any case, I am not aware of anyone approaching the issue of latent zero-day vulnerabilities, *per se,* with these techniques, techniques that software engineering has adapted from the biology world. Certainly, papers as early as Ozment & Schechter's "Milk or Wine: Does Software Security Improve with Age?" [9] looked at the declining rate of flaw finding within a software project under consistent management (OpenBSD), but that is subtly different and, in any case, should probably be evaluated as an example of removal-capture rather than an example of capture-recapture.

It seems to me that the most straightforward way to make a first quantitative effort here is to employ three or more independent penetration tests against the same target. Or have your software looked over by three or more firms offering static analysis. Scott & Wohlin's case study [10] with the KDE Open Source project and UIQ Technology might be worth copying.

Perhaps we can take a large body of code and look at the patches that have been issued against it over time. If you take a patch as a marker for a previously undiscovered flaw, then the rate at which patches issue is a removal-capture process. Were that process to maintain a relatively constant hum, then it might imply that software flaws are indeed dense—too dense to diminish with removals. Of course, patches for a commercial software system are not necessarily unitary—one apparent patch may actually fix several flaws. Rescorla concluded that fixing without disclosure is better than fixing with disclosure (and thus was "an advantage for closed source over open source"), but such a policy certainly doesn't help us do quantitative research with real data.

There is something here to work with for those who test or who can closely observe those who do. Be in touch; I'd like to work with you.

References

[1] Multiple articles in *Statistical Science,* vol. 9 no. 4, 1994.

[2] N. McKeganey et al., "Female Streetworking Prostitution and HIV Infection in Glasgow," *British Medical Journal* (1992), vol. 305, pp. 801–804.

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

[4] E. Rescorla, "Is Finding Security Holes a Good Idea?" Workshop on the Economics of Information Security, 2004.

[5] S. Clark, S. Collis, M. Blaze, and J. M. Smith, "Moving Target: Security and Rapid-Release in Firefox," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security* (ACM, 2014) pp. 1256–1266: dl.acm.org/citation.cfm?id=2660320.

[6] S. A. Vander Wiel and L. G. Votta, "Assessing Software Designs Using Capture-Recapture Methods," *IEEE Transactions on Software Engineering (1993),* vol. 19, no. 11, pp. 1045–1054.

[7] http://www.underhanded-c.org/: "The goal of the contest is to write code that is as readable, clear, innocent and straightforward as possible, and yet it must fail to perform its apparent function. To be more specific, it should do something subtly evil."

[8] H. Petersson, T. Thelin, P. Runeson, and C. Wohlin, "Capture-Recapture in Software Inspections after 10 Years of Research," *Journal of Systems and Software* (July 2004), vol. 72, no. 2, pp. 249–264.

[9] A. Ozment and S. E. Schechter, "Milk or Wine: Does Software Security Improve with Age?" in *Proceedings of the 15th Conference on USENIX Security Symposium* (2006), pp. 93–104.

[10] H. Scott and C. Wohlin, "Capture-Recapture in Software Unit Testing—A Case Study," Blekinge Institute of Technology, 2004: www.wohlin.eu/esem08-1.pdf.

usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Become a USENIX Supporter and Reach Your Target Audience

The USENIX Association welcomes industrial sponsorship and offers custom packages to help you promote your organization, programs, and products to our membership and conference attendees.

Whether you are interested in sales, recruiting top talent, or branding to a highly targeted audience, we offer key outreach for our sponsors. To learn more about becoming a USENIX Supporter, as well as our multiple conference sponsorship packages, please contact sponsorship@usenix.org.

Your support of the USENIX Association furthers our goal of fostering technical excellence and innovation in neutral forums. Sponsorship of USENIX keeps our conferences affordable for all and supports scholarships for students, equal representation of women and minorities in the computing research community, and the development of open source technology.

Learn more at:
www.usenix.org/supporter