Fabio Massacci
Associate Editor in Chief

Pseudo Ground-Truth Generators and Large-Scale Studies

You have found out that 123,456,789 Android/Java/C applications have some vulnerability, and you can automatically fix 123,454,321 of them (and not one fewer). Before you rush to a top security conference, let me cast the seed of doubt: "How can you be sure?"

Large-scale studies are becoming more common and, increasingly, require a macho showing of gazillions of analyzed malware samples, Android applications, and software libraries—you name it—that lazy or inept developers, administrators, and users aren't doing enough about. Isn't this a good trend toward a quantitative security science? Well, read again Herley and van Oorschot[1] on the perils of unstated assumptions and uncontextualized solutions.

So, what's the problem? If you must scan a gazillion Xs to make it to the top conferences, then how can you be sure that 93% (or 57%, or 21%, and so on) of them have a vulnerability, can be fixed/updated, and so on?

The short answer is "You can't." The long answer is that you cannot manually and thoroughly check a gazillion Xs, so you need a tool to check for correctness, and, therefore, you must use an automatic *ground-truth generator* (*GTG*).

Most papers give us the nice part of the story, which I illustrate in Figure 1: Pipe your Xs (Android apps, websites, libraries, malware samples, and so on) through the first GTG box, which determines that some Xs have a problem (for example, Xs that are Android applications with a vulnerability, malware samples that escape detection, and so on). Then, pipe all found unsecure Xs through another box, which fixes them into secure Ys (or more precisely detects them, and so forth). This box is a "diagnose and fix" tool (D&FT). Finally, pass the surfacing Ys through another box, the second GTG, which checks that they are OK. The outcome of an algorithm has become the "ground truth": whether you needed fixing beforehand or whether you are "fixed" afterward.

Lather, rinse, and repeat for a gazillion times, and you have an A* paper with 123,454,321 (and not one fewer) automatically fixed you-name-its. How can we be sure?

In theory, the tool chain could make us sure, but, in practice, that's a superbad assumption.[1] The GTG is a tool, and, as with any tool, it makes errors by either bugs or abstraction, and these accumulate.

Those who studied cryptographic random number generators might remember von Neumann's quote that "Anyone who attempts to generate random numbers by deterministic means is, of course, living in a state of sin." Let me give similar advice: anyone who attempts to generate ground truths by algorithmic means is, of course, living in a state of sin.

Don't get me wrong: I also use GTGs,[4,5] and I believe they are the only way to scale. Still, while there are great papers,[2] there are also sinners.[3] What is (badly) wrong is that we don't report how error prone they can be, and we set forth our own generated pseudo-truth as the "truth" or a "proof." (To avoid this pitfall, Dashevskyi et al.[5] deliberately used the word "evidence.")

However, in the same way that pseudorandom number generators can be useful, so can a pseudo-GTG. We just need a (scientific) way to assess the errors of our pseudo-GTGs and understand how these they are propagated across our pipeline.

Let me illustrate the problems with some papers. I read the 16,837-Android-apps paper[3] at the Association for Computing Machinery

Digital Object Identifier 10.1109/MSEC.2021.3137674
Date of current version: 21 March 2022

Conference on Computer and Communications Security 2017, and I was fascinated: all if these 16,837 vulnerable libraries (and not one less) could be fixed just by updating to the next version, an "updatability rate" of more than 93%. Another group identified 412,288 Java dependencies (and not one less) that were not used (a "bloated dependencies" study) and could be automatically and safely removed, thus decreasing the attack surface of a library.[6]

A moment of reckoning arrived for me at the IEEE European Symposium on Security and Privacy 2019 in Stockholm. A Ph.D. student presented a framework for the dynamic testing of Android applications,[2] which essentially revisited the work of Derr et al.[3] with a masterpiece of understatement: "Prior reported updatability rate is, under real conditions, overestimated." Well, inside the paper, she showed that the updatability may be reduced from 93% to 47%. My summary would have been, "The prior report[3] is utterly misleading."

At a Dagstuhl seminar, I asked one of the authors of the 16,837-apps paper,[3] "How can we be sure they are 16,837 (and not one less)?" I wasn't asking for a retraction; I really wanted to know if they had thought about the problem after the work by Huang et al.[2] He didn't have an answer besides "Our tool chain says so."

A few weeks ago, the same colleague of the "bloated dependencies"[6] sent me a paper[7] in which the authors showed that, to increase diversity (and, thus, robustness against attacks), one can automatically replace a library with a different version of the same library: 169 libraries can be replaced, not by just one different version, but interchangeably with 15 of them, for a total of 2,535 combinations. Again, I asked, "How can we be sure?"; again, the answer was, "Our tool chain says so."

Can we do something better? There are techniques to compute some errors, such as using Agresti–Coull manual interval analysis for the probability error of a proportion (the proportion of correct answers of each pseudo-GTG).

Let me illustrate it with my own large-scale study. At some point, we wanted to replicate the milk-or-wine study[8] and check whether all browser vulnerabilities were foundational (i.e., present since the very beginning of a software commit history). Ozment and Schechter[8] or Meeneley and Williams[9] lived in a different era and could go manual, but, when I, Dashevskyi, and Nguyen started the research behind our 2016 article,[4] it was already the dawn of the machos . . . so we built a structure similar to Figure 1, writing programs, integrating industry databases, scanning gazillions of commits in minutes over open source repositories and—"Hooray! Chrome's vulnerabilities are essentially foundational."

However, you wouldn't find this "finding" in our papers on all major browsers[4] and Java libraries.[5] What happened? We were lucky enough that a stubborn reviewer of our journal submission asked us, "How can you be sure?"

Digital Object Identifier 10.1109/MSEC.2021.3130867

Therefore, we went manual for a sufficiently significant sample for Agresti–Coull. Oops. We found out that the reliable industry sources we used to determine whether a version was vulnerable weren't that reliable. In many allegedly vulnerable versions, the code fragment responsible for the vulnerability wasn't in the version's codebase. It is just so easy (and way safer) for a security researcher reporting vulnerabilities to say, "Version X is vulnerable, and so are all its previous versions." Our paper mutated between revisions.

Are we heading toward the same crisis well described in Florencio and Herley's famous paper[10] on sex, lies, and cybercrime surveys—this time for large-scale studies? Look again at Figure 1. Subtle and yet practical issues lurk into the shadows:

1. The GTG correctly identified a secure X as secure, but you don't make an A* conference paper by saying, "Hey, 84% of Xs aren't obviously vulnerable, so you shouldn't worry." Free advice to Ph.D. students: academic reviewers don't like it, and it takes ages to publish a paper. We tried that; don't do it. Second thoughts: companies do like it, and you may end up in an industry standard. We tried that; do it.

More serious problems accumulate as we move along the pipeline:

1. The vulnerable individual X was actually correctly fixed/detected by the D&FT into Y. This is the good case and, alas, the only one.
2. The diagnosis was correct, and the individual in question might have well been "fixed," but this fix mangled it. However, from the perspective of the second GTG, all was well. Given the evidence of Huang et al.[2] and the comment by developers reported by Soto-Valero et al.,[6] I suspect this is the most frequent case for all methods published so far that claimed to have a "proof."
3. The first GTG might have actually been wrong (the perils of abstraction go both ways[1]) and sent a perfectly secure individual to the D&FT, which hacked the "malicious" parts and made another mangled to satisfy the second GTG. I also suspect that this frequently happens the farther the GTG is from executing an actual exploit. This was our experience[4] on versions "claimed to be vulnerable."

We take for granted that the D&FT always fixes the result. We have no warranty about it.

4. The first GTG was right and sent a vulnerable individual to the surgery of the D&FT, which removed the "wrong" parts and, thus, not only made a mangled but also a vulnerable one, too; that, however, succeeded for the second GTG (wrong in the wrong direction).

To understand the last point, Table 1 shows, side by side, a "correct" fix generated when Cuong Quang, a Ph.D. student at the Technical University of

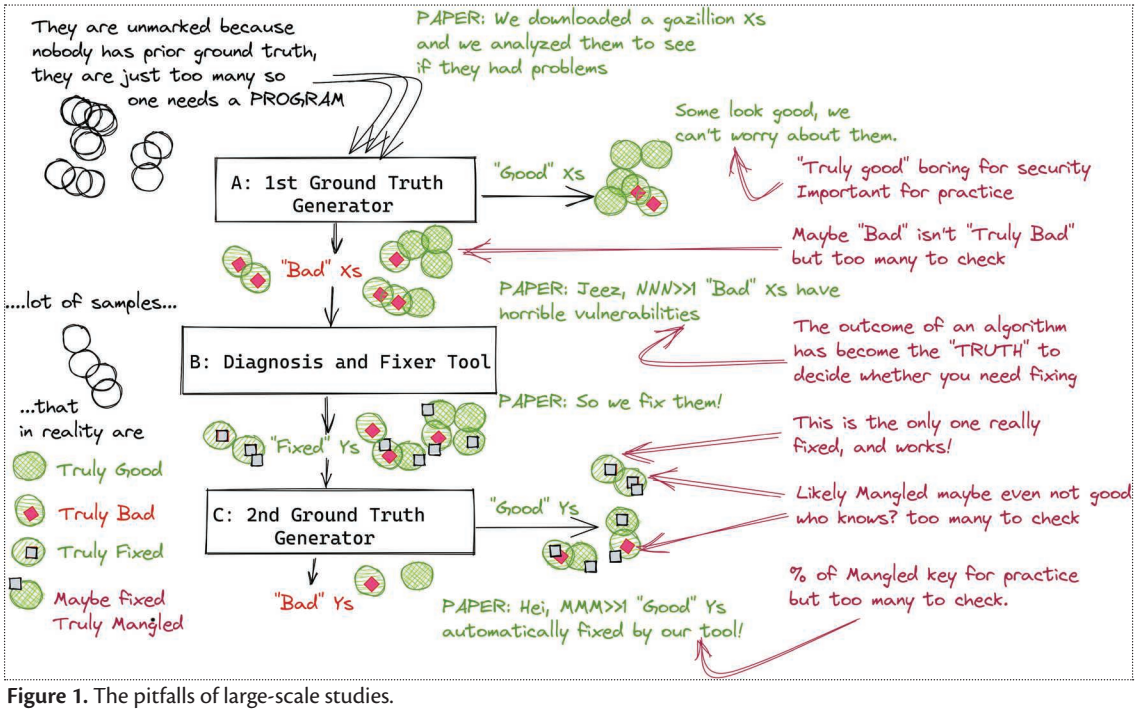

Figure 1. The pitfalls of large-scale studies.

Table 1. Automated program repair for vulnerabilities.

Developer's fix of CVE-2018-1324	TBar's "correct" fix	Kali's "correct" fix
- for (int i = 0; i < this.rcount; i++) { + for (long i = 0; i < this.rcount; i++) {	- for (int i = 0; i < this.rcount; i++) { + for (int i = 0; i == this.rcount; i++) {	+ if (true) + return;

The table reports the fix according to two state-of-the-art automated program repair tools for Java programs, TBar[11] and Kali.[12] The vulnerability is CVE-2018-1324 for Apache Commons. The fix is "correct" according to the current evaluation practices, as it fixes all regression test cases, including those showing that there was a vulnerability. (In this case, the GTG is the tool test suite from Maven).

Hamburg Harburg applied state-of-the-art "automated program repair" (APR) tools[11,12] to a real-life vulnerable library and compared it to the fix by the developer. The problem is not that APR tools are wrong; the problem is that the state of the art labels those types of answers as "correct" when ranking tools.

We must learn to be skeptical of all studies that report no errors, no uncertainty, and no manual (significant) validation and, yet, large numbers precise to the single digit. There is nothing bad about having big errors. We, as reviewers, must be willing to accept studies where such errors are reported—and, if you have one such paper, consider sending it to us. Even in the government's approved final title card in *Not One Less*, only 15% of the rural children make it back to school.[13]

Unless we start taking action as a community, there will be an increasingly widening gap between the claims in papers (93% in Derr et al.[3]—one's algorithm failure is as rare as four heads in a row) and practical reality (47% in Huang et al.[2]—real failure is as common as a single coin toss). Let us know what you think. ■

Acknowledgments

I am thankful to Cuong Bui Quang for showing me the example mentioned in Table 1, Antonino Sabetta for many discussions on what really matters in industry, and Sean Peisert and Paul van Oorschot for helping me to sharpen my point. This work was supported by the European Union H2020 program under grant 952647 (AssureMOSS). Any opinion (possibly right only 47–93% of the time) is, of course, mine.

References

1. C. Herley and P. C. Van Oorschot, "Science of security: Combining theory and measurement to reflect the observable," *IEEE Security Privacy*, vol. 16, no. 1, pp. 12–22, 2018, doi: 10.1109/MSP.2018.1331028.
2. J. Huang, N. Borges, S. Bugiel, and M. Backes, "Up-to-crash: Evaluating third-party library updatability on Android," in *Proc. IEEE Eur. Symp. Security Privacy (EuroS&P)*, 2019, pp. 15–30, doi: 10.1109/EuroSP.2019.00012.
3. E. Derr, S. Bugiel, S. Fahl, Y. Acar, and M. Backes, "Keep me updated: An empirical study of third-party library updatability on Android," in *Proc. ACM Conf. Comput. Commun. Security*, 2017, pp. 2187–2200, doi: 10.1145/3133956.3134059.
4. V. H. Nguyen, S. Dashevskyi, and F. Massacci, "An automatic method for assessing the versions affected by a vulnerability," *Empirical Softw. Eng.*, vol. 21, no. 6, pp. 2268–2297, 2016, doi: 10.1007/s10664-015-9408-2.
5. S. Dashevskyi, A. D. Brucker, and F. Massacci, "A screening test for disclosed vulnerabilities in FOSS components," *IEEE Trans. Softw. Eng.*, vol. 45, no. 10, pp. 945–966, 2018, doi: 10.1109/TSE.2018.2816033.
6. C. Soto-Valero, N. Harrand, M. Monperrus, and B. Baudry, "A comprehensive study of bloated dependencies in the Maven ecosystem," *Empirical Softw. Eng.*, vol. 26, no. 3, pp. 1–44, 2021, doi: 10.1007/s10664-020-09914-8.
7. N. Harrand, T. Durieux, D. Broman, and B. Baudry, "Automatic diversity in the software supply chain," 2021, arXiv:2111.03154.
8. A. Ozment and S. E. Schechter, "Milk or wine: Does software security improve with age?" in *Proc. 15th USENIX Security Symp.*, 2006, vol. 6, pp. 10–5555.
9. A. Meneely and L. Williams, "Secure open source collaboration: An empirical study of Linus' law," in *Proc. ACM Conf. Comput. Commun. Security*, 2009, pp. 453–462, doi: 10.1145/1653662.1653717.
10. D. Florêncio and C. Herley, "Sex, lies and cyber-crime surveys," in *Economics of Information Security and Privacy III*, B. Schneier, Ed. New York, NY, USA: Springer Science & Business Media, 2013, pp. 35–53.
11. Z. Qi, F. Long, S. Achour, and M. Rinard, "An analysis of patch plausibility and correctness for generate-and-validate patch generation systems," in *Proc. ACM Int. Symp. Softw. Testing Analysis*, 2015, pp. 24–36, doi: 10.1145/2771783.2771791.
12. K. Liu, A. Koyuncu, D. Kim, and T. F. Bissyandé, "TBar: Revisiting template-based automated program repair," in *Proc. ACM Int. Symp. Softw. Testing Analysis*, 2019, pp. 31–42, doi: 10.1145/3293882.3330577.
13. Z. Yimou, "Not One Less," Wikipedia, 1999. https://en.wikipedia.org/wiki/Not_One_Less