Almost Too Big to Fail

DAN GEER AND JOSHUA CORMAN

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

Joshua Corman is the chief technology officer for Sonatype. Previously, Corman served as a security researcher and strategist at Akamai Technologies, The 451 Group, and IBM Internet Security Systems. A respected innovator, he co-founded Rugged Software and I Am the Cavalry to encourage new security approaches in response to the world's increasing dependence on digital infrastructure. He is also an adjunct faculty for Carnegie Mellon's Heinze College, IANS Research, and a Fellow at the Ponemon Institute. Josh received his bachelor's degree in philosophy, graduating summa cum laude, from the University of New Hampshire. joshcorman@gmail.com

Both dependence on open source and adversary activity around open source are widespread and growing, but the dynamic pattern of use requires new means to estimate if not bound the security implications. In April and May 2014, every security writer has talked about whether it is indeed true that with enough eyeballs, all bugs are shallow. We won't revisit that topic because there may be no minds left to change. Unarguably:

- Dependence on open source is growing in volume and variety.
- Adversary interest tracks installed base.
- Multiple levels of abstraction add noise to remediation needs.

We begin with two open source examples.

Apache Struts CVE-2013-2251, July 6, 2013 - CVSS v2 9.3
Apache Struts is one of the most popular and widely depended upon open source projects in the world. As such, when this highly exploitable vulnerability was discovered, it was promptly used to compromise large swaths of the financial services sector. While Heartbleed (see below) got full media frenzy, many affected by 2013-2251 learned of the problem from FBI victim notifications under 42 U.S.C. § 10607. The FS-ISAC issued guidance [1] telling institutions (read, victims) to scrutinize the security of third-party and open source components throughout their life cycle of use. It is not noteworthy that an open source project could have a severe vulnerability; what *is* of note is that this flaw went undetected for at least seven years (if not a lot longer from WebWork 2/pre-Struts 2 code base)—an existence proof that well-vetted code still needs a backup plan.

OpenSSL (Heartbleed) CVE-2014-0160, April 7, 2014 - CVSS v2 5.0
The Heartbleed vulnerability in OpenSSL garnered tremendous media and attacker activity this past April. While only scored with a CVSS of 5.0, it is a "5 with the power of a 10" since sniffing usernames, passwords, and SSL Certificates provides stepping stones to far greater impact. In contrast to the Struts bug above, this flaw was introduced only two years prior, but it, too, went unnoticed by many eyeballs—it was found by bench analysis [2].

Dependence on Open Source Is Growing

Sonatype, home to author Corman, serves as custodian to Central Repository, the largest parts warehouse in the world for open source components. At the macro level, open source consumption is exploding in Web applications, mobility, cloud, etc., driven in part by increasingly favorable economics. Even (risk averse, highly regulated) government and financial sectors, which previously resisted "code of unknown origin/quality/security," have begun relaxing their resistance. According to both Gartner surveys and Sonatype application analysis, 90+% of modern applications are not so much written as assembled from third-party building blocks. It is the open source building blocks that are taking the field, and not just for commodity applications (see Figure 1).

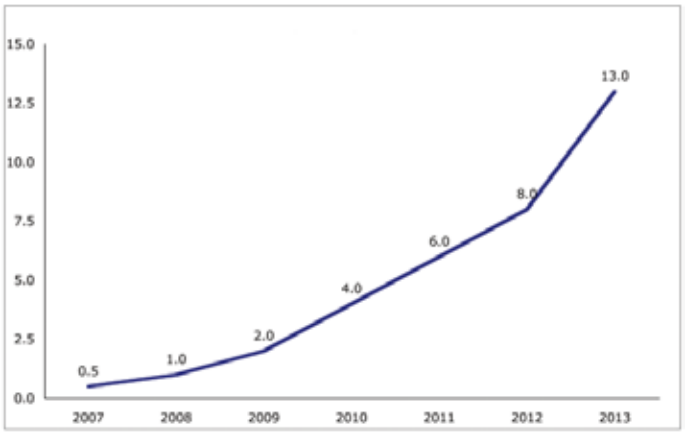

Figure 1: Open source downloads per year measured in billions

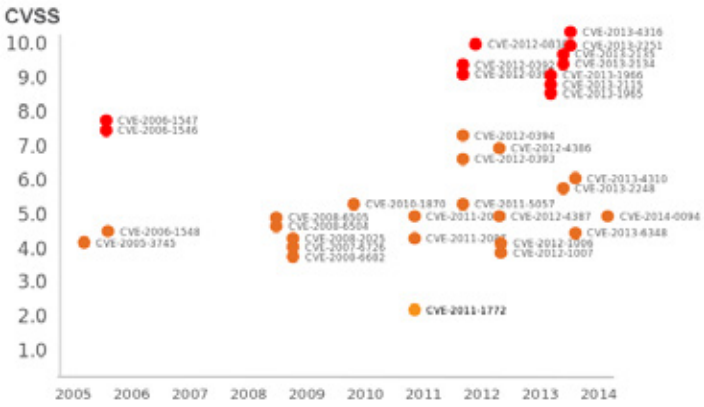

Figure 2: Graphing the CVSS severity (1–10) for disclosed Struts vulnerabilities against the year shows generally increasing severity levels.

Adversary Interest in Open Source Is Growing

Adversary interest tracks component prevalence. The prevalence of open source has grown, ergo so has adversary interest [3]. There are several equivalent ways to characterize that:

- Payoff: "That's where the money is."
- Cost-effective leverage: Unless you are engaged in one-off targeting, you go after the components that are most depended upon (Struts, OpenSSL, etc.).
- Accessibility: Obscurity may occasionally contribute to security, but there is nothing obscure about an open source code pool.

Figure 2 shows the pattern of vulnerability disclosure in the Apache Struts project; the vertical axis shows CVSS severity against the horizontal showing calendar time.

While author Geer has written elsewhere [4] about how CVSS scores are *not* the way to steer remediation efforts, Figure 2 does confirm that there is a mounting interest in cataloging open source flaws. (See also author Corman's "HDMoore's Law" [5].)

Can We Characterize Flaw Response?

Yes, Virginia, all software has flaws, but one might ask whether we avoid "known bad components" when assembling deliverable code? Not always; consider:

"Bouncy Castle" CVE-2007-6721, November 10, 2007
CVSS v2 10

The "Legion of the Bouncy Castle Java Cryptography APIs" had a CVSS worst-case scenario fixed in April of 2008—more than six years ago. While 2007-6721 is a severe security flaw in a security-sensitive project, nevertheless the unrepaired, vulnerable version was requested from Central Repository 4,000 times in 2013. One can assume it was used in security-related applications/products, perhaps multiple applications per download instance.

Similar (disappointing) consumption patterns exist for Struts. Outside of *CVE-2013-2251* compromised organizations, still-vulnerable versions of Struts 2 continue to remain popular. Worse, Struts version 1-related artifacts still had over a million downloads in 2013, despite its April 5, 2013 official End of Life. In other words, finding and fixing serious flaws in open source does not mean that the repaired versions are the ones that are used. Is this an awareness problem, or is it something else?

Readers will recall that Availability (A) is calculated as

$$A = \frac{MTBF}{MTBF + MTTR}$$

where MTTR is Mean Time To Repair and MTBF is Mean Time Between Failures. Availability is thus perfect (100%) if either the item never fails (MTBF goes to infinity) or the item enjoys instant recovery (MTTR goes to 0). This is where a distinction between open and closed source may be operationally relevant: If the MTBF is a constant, then MTTR is what matters. The 2013 Coverity Scan Report [6] showed comparable defect rates between open and closed source projects (with a slight quality advantage for open source projects). If project sizes are also comparable, then MTBF between open and closed source would likewise be comparable.

We have less data on MTTR, whether for closed or open source, but it is our educated guess that (once fixed) open source project repairs are *available* earlier than closed source projects because the latter will have additional packaging and deployment steps. Open source projects are not responsible for deployment of fixes, only the availability of fixes, and, even then, there is no forcing function for making fixes available. In a sense, Heartbleed was a blessing; it showed us just how widespread one error can be deployed and just how much widespread use led users to assume that it must have been thoroughly scrubbed by somebody else by now.

But to base our discussion on knowledge rather than educated guesses, Sonatype has begun an analysis of the "project integrity" of the open source codebases it hosts. One focus will, in fact, be MTTR. It is central to the open source domain because there are unobvious transitive dependencies between and among open source components. An early analysis of open source projects with already identified vulnerable dependencies revealed some troubling behavior. Direct (aka "1-hop") vulnerable component dependencies were only remediated 41% of the time. Put differently, more than half (59%) of the vulnerable base components remain unrepaired. Folding multiple components into your projects means inheriting not just the components' functionality but also their (largely unrepaired) flaws. For the 41% that were fixed at all, the MTTR was 390 days (median 265 days). Filtering for just CVSS 10s brought the mean of this subset down to 224 days. And this is just for 1-hop dependencies—there is as yet no mechanism to cause remediated flaws to flow automatically through the dependency graph, and there may never be.

Making Remediation Possible

In closed source development domains, the command structure will know who uses what and can thus ascertain what code trees have to be rippled when a common component is revised. This is not the case with open source, nor will it be. As Heartbleed made clear, open source is in home electronics, medical devices, industrial controls, etc. The more widespread the use of a particular open source library, the more common mode failure among otherwise unrelated product spaces becomes. An auto manufacturer can recall a particular model, and know that only that model has the faulty component. There is no feasible equivalent for an open source library. We thus suggest that, just as a jar of pickles on the grocery shelf must list its ingredients, products and services that are assembled from open source components need to provide a bill of materials so that when an open source component has a vulnerability, downstream users can tell whether they are affected and whether a particular remediation is one they need to consume (directly or indirectly). Ingredients lists would serve as a framework both for remediation and for further work in security metrics.

To emphasize the concreteness of these issues, embedded systems are largely assembled from open source components, have no field upgrade path once deployed, and had build environments that were not coordinated with source code control. We have work to do.

References

[1] Financial Services Information Sharing and Analysis Center, "Appropriate Software Security Control Types for Third Party Service and Product Providers": docs.ismgcorp.com/files/external/WP_FSISAC_Third_Party_Software_Security_Working_Group.pdf.

[2] F. Berkman, "Researcher Who Discovered Heartbleed Bug Donates $15K Reward," *The Daily Dot*: www.dailydot.com/news/heartbleed-neel-mehta-freedom-press-foundation-encryption.

[3] S. Rosenblatt, "Heartburn from Heartbleed Forces Wide-Ranging Rethink in Open Source World," CNET: www.cnet.com/news/heartburn-from-heartbleed-forces-wide-ranging-rethink-in-open-source-world.

[4] D. Geer and M. Roytman, "Measuring vs. Modeling," USENIX *;login:*, vol. 38, no. 6 (December 2013): geer.tinho.net/fgm/fgm.geer.1312.pdf.

[5] J. Corman, "Intro to HDMoore's Law": blog.cognitivedissidents.com/2011/11/01/intro-to-hdmoores-law/.

[6] Coverity Scan, 2013 Open Source Report: softwareintegrity.coverity.com/rs/coverity/images/2013-Coverity-Scan-Report.pdf.