Looking for Lacunae in Bitcoin Core's Fuzzing Efforts

Alex Groce
Northern Arizona University
United States

ABSTRACT

Bitcoin is one of the most prominent distributed software systems in the world, and a key part of a potentially revolutionary new form of financial tool, cryptocurrency. At heart, Bitcoin exists as a set of nodes running an implementation of the Bitcoin protocol. This paper describes an effort to investigate and enhance the effectiveness of the Bitcoin Core implementation fuzzing effort. The effort initially began as a query about how to escape *saturation* in the fuzzing effort, but developed into a more general exploration once it was determined that saturation was largely illusory, a byproduct of the (then) fuzzing configuration. This paper reports the process and outcomes of the two-week focused effort that emerged from that initial contact between Chaincode labs and academic researchers. That effort found no smoking guns indicating major test/fuzz weaknesses. However, it produced a large number of additional fuzz corpus entries to add to the Bitcoin QA assets, clarified some long-standing problems in OSSFuzz triage, increased the set of documented fuzzers used in Bitcoin Core testing, and ran the first mutation analysis of Bitcoin Core code, revealing opportunities for further improvement. We contrast the Bitcoin Core transaction verification testing with similar tests for the most popular Ethereum and dogecoin implementations. This paper provides an overview of the challenges involved in improving testing infrastructure, processess, and documentation for a highly visible open source target system, from both the state-of-the-art research perspective and the practical engineering perspective.

CCS CONCEPTS

• **Software and its engineering** → **Dynamic analysis**; **Software testing and debugging**.

KEYWORDS

fuzzing, saturation, test diversity, mutation analysis

ACM Reference Format:
Alex Groce. 2021. Looking for Lacunae in Bitcoin Core's Fuzzing Efforts. In *Proceedings of the 2021 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward! '21), October 20–22, 2021, Chicago, IL, USA.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3486607.3486772

Onward! '21, October 20–22, 2021, Chicago, IL, USA

ACM ISBN 978-1-4503-9110-8/21/10…$15.00
https://doi.org/10.1145/3486607.3486772

1 INTRODUCTION

Bitcoin is the most popular cryptocurrency, and one of the most visible "new" systems based on software to rise to prominence in the last decade. As we write, while volatile, Bitcoin consistently has a market cap of over half a trillion dollars since January of 2021. Due to its distributed, decentralized nature, Bitcoin in some sense is the sum of the operations of the code executed by many independent Bitcoin nodes, especially nodes that mine cryptocurrency. Bitcoin Core (https://github.com/Bitcoin/Bitcoin) is by far the most popular implementation, and serves as a reference for all other implementations. To a significant degreee, the code of Bitcoin Core *is* Bitcoin. The main Bitcoin Core repo on GitHub has over 57,000 stars, and has been forked more than 30,000 times.

Because of its fame and the high monetary value of Bitcoins, the Bitcoin protocol and its implementations are a high-value target for hackers (or even nation states interested in controlling cryptocurrency developments).

2 INITIAL CONTACT AND THE PROBLEM OF SATURATION

Chaincode labs (https://chaincode.com/) is a private R&D center based in Manhattan that exists solely to support and develop Bitcoin. In March of 2021, Adam Jonas, the head of special projects at Chaincode, contacted the first author to discuss determining a strategy to improve the fuzzing of Bitcoin Core. In particular, at the time, it seemed that the fuzzing was "stuck": neither code coverage nor found bugs were increasing with additional fuzzing time. After some discussion, an 80 hour effort was determined as a reasonable scope for an external, research-oriented, look at the fuzzing effort. Before that effort, conducted over the summer of 2021, began, the problem of saturation resolved itself. Nonetheless, the issue that drove the initial desire for a researcher investigation is well worth examining. Moreover, understanding why Bitcoin Core fuzzing was, temporarily but not fundamentally, saturated, may be useful to help other fuzzing campaigns avoid the same false saturation problem.

Saturation, as defined in the blog post (https://blog.regehr.org/archives/1796) that brought Chaincode labs to the first author, is when "We apply a fuzzer to some non-trivial system... initially it finds a lot of bugs... [but] the number of new bugs found by the fuzzer drops off, eventually approaching zero." That is, at first a fuzzer applied to a system will tend to continuously, sometimes impressively, increase both coverage and discovery of previously-unknown bugs. But, at some point, these bugs are known (and often fixed) and the fuzzer stops producing new bugs. Code and behavioral coverage seems to be *saturated*.

The underlying reason for saturation is that any fuzzer (or other test generator) explores a space of generated tests according to some, perhaps very complex, probability distribution. Some bugs lie in the high-probabiliity portion of this space, and other bugs like in very low probability (or in some cases, zero probability)

parts of the space. Unsurprisingly, eventually the high probability space is well-explored, and the remaining bugs are found only very infrequently, if ever. The underlying empirical facts are cruel, as noted by Böhme and Falk: "[W]ith twice the machines, we can find *all known* bugs in half the time. Yet, finding linearly *more* bugs in the same time requires exponentially more machines" [1].

Chaincode labs saw that code coverage and bugs were not increasing in their fuzzer runs, and wanted to break out of the saturation trap. The blog post suggested several methods for doing just that, and this paper explores some of them. Note that even though the fuzzing was not saturated, the methods for escaping saturation are also useful things to try in any fuzzing effort where finding as many bugs as possible is actually important.

2.1 We Hold the World Ransom For... ONE MILLION FUZZER ITERATIONS

3 ADDING FUZZER DIVERSITY: USING ECLIPSER AND TRYING ENSEMBLE FUZZING

Eclipser [2] is a fuzzer that combines afl-like[1] coverage-driven mutation-based fuzzing with a scalable form of grey-box concolic testing.

4 SIDE ISSUES: TRIAGE, SPACE USAGE, AND FRUSTRATING BUGS

5 TRYING SWARM FUZZING

DeepState [3] provides strong support for swarm fuzzing. However, while there are similarities to the bitcoin core developed infrastucture for fuzz targets and the DeepState API, re-writing the fuzz targets to use DeepState is too large an effort for the likely payoff at this time, even though DeepState would also provide additional fuzzer diversity.

6 MUTATION ANALYSIS

We used the universal mutator (https://github.com/agroce/universalmutator) [4] to mutate the code.

6.1 Comparison with Geth and Dogecoin

7 CONCLUSIONS

REFERENCES

[1] Marcel Böhme and Brandon Falk. 2020. Fuzzing: On the Exponential Cost of Vulnerability Discovery. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Virtual Event, USA) *(ESEC/FSE 2020)*. Association for Computing Machinery, New York, NY, USA, 713–724. https://doi.org/10.1145/3368089.3409729

[2] Jaeseung Choi, Joonun Jang, Choongwoo Han, and Sang Kil Cha. 2019. Grey-box Concolic Testing on Binary Code. In *Proceedings of the International Conference on Software Engineering*. 736–747.

[3] Peter Goodman and Alex Groce. 2018. DeepState: Symbolic unit testing for C and C++. In *NDSS Workshop on Binary Analysis Research*.

[4] Alex Groce, Josie Holmes, Darko Marinov, August Shi, and Lingming Zhang. 2018. An Extensible, Regular-expression-based Tool for Multi-language Mutant Generation. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings* (Gothenburg, Sweden) *(ICSE '18)*. ACM, New York, NY, USA, 25–28. https://doi.org/10.1145/3183440.3183485

[1] In fact, the latest version of Eclipser simply uses afl for non concolic fuzzing.

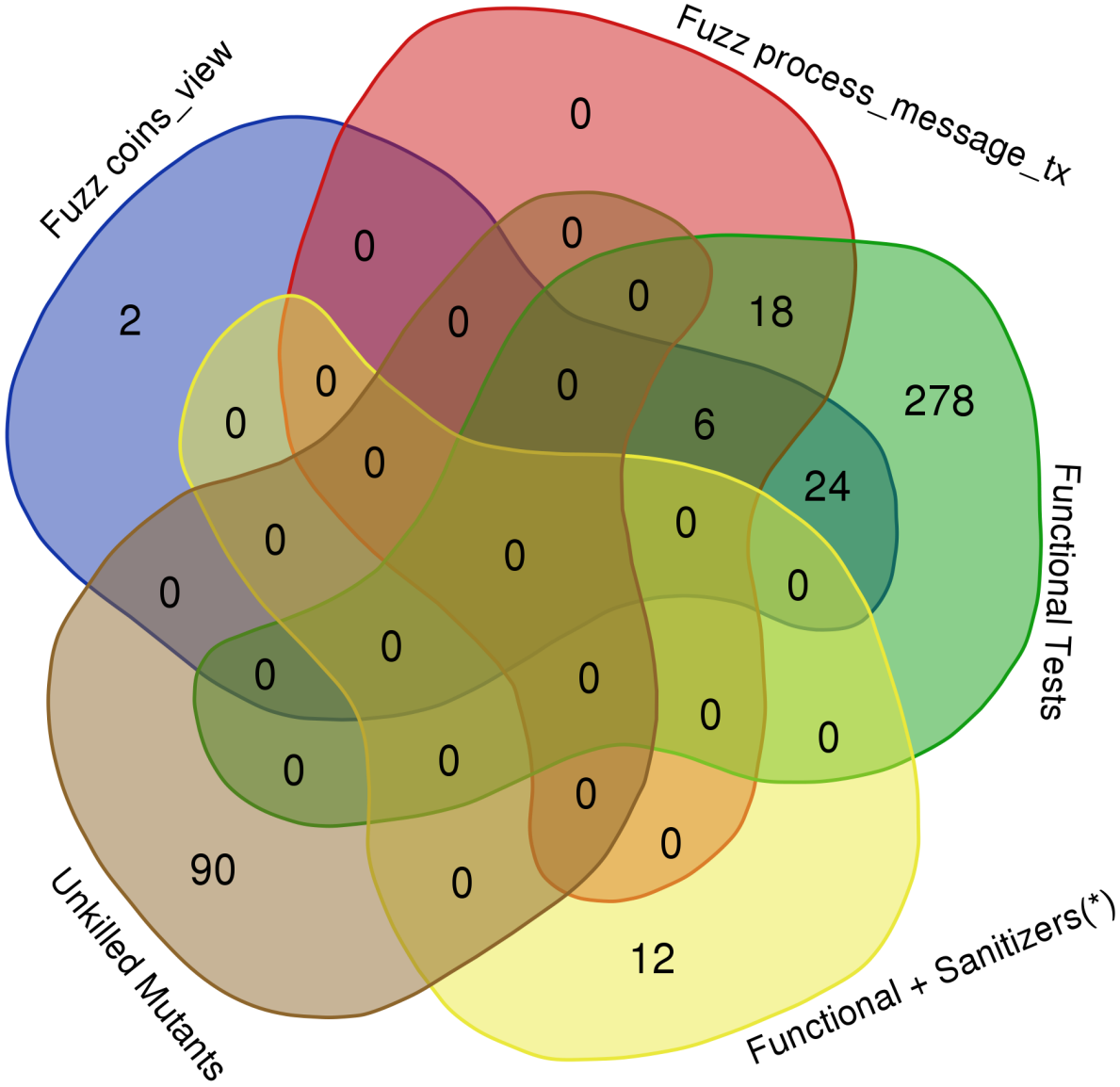

Figure 1: The Gaping Maw of Cthulhu