Volume 123, Article No. 123005 (2018) https://doi.org/10.6028/jres.123.005
Journal of Research of National Institute of Standards and Technology

A Software Assurance Reference Dataset: Thousands of Programs With Known Bugs

Paul E. Black

National Institute of Standards and Technology,
Gaithersburg, MD 20899

paul.black@nist.gov

Data DOI: https://doi.org/10.18434/T4/1433084

Key words: cybersecurity; software assurance; software quality; static analysis.

Accepted: April 6, 2018

Published: April 16, 2018

https://doi.org/10.6028/jres.123.005

1. Summary

The Software Assurance Reference Dataset (SARD) [1] is a growing collection of over 170 000 programs with precisely located bugs. The programs are in C, C++, Java[1], PHP, and C# and cover more than 150 classes of weaknesses, such as SQL injection, cross-site scripting (XSS), buffer overflow, and use of a broken cryptographic algorithm. Most are automatically generated synthetic programs, each a few pages of code long, but there are also over 7000 full-sized applications. In addition, SARD has production code and has hundreds of cases written by hand. The code is typical quality. It is neither pristine nor obfuscated. Many cases have corresponding "good" cases, in which weaknesses are fixed, to test for false positives.

The SARD web interface allows users to browse test cases and test suites or search for test cases by programming language, weakness type, file name, size, words in the description, and several other criteria. The user can select and download any or all of the resulting cases.

2. Data Specifications

NIST Operating Unit	Information Technology Laboratory, Software and Systems Division, Software Quality Group
Format	ASCII source code
Instrument	N/A
Spatial or Temporal Elements	N/A
Data Dictionary	Download file format: https://samate.nist.gov/SARD/resources/sard_schema.xsd
Accessibility	Publicly available.
License	Varies. Many test cases are public domain.

[1]Certain trade names and company products are mentioned in the text or identified for adequate documentation. In no case does such identification imply recommendation or endorsement by the National Institute of Standards and Technology (NIST), nor does it imply that the products are necessarily the best available for the purpose.

How to cite this article:
Black PE (2018) A Software Assurance Reference Dataset: Thousands of Programs With Known Bugs.
J Res Natl Inst Stan 123:123005. https://doi.org/10.6028/jres.123.005.

Each test case has metadata to describe it. Most bugs or weaknesses are recorded in metadata. Weaknesses are classified using the Common Weakness Enumeration (CWE) [2] ID and name. We plan to add their Bugs Framework (BF) [3] class and attributes.

Thousands of cases are synthetic, that is, generated by programs. These generators produce variants of a basic weakness in different *code complexities*, e.g., wrapping the weakness in assorted conditionals and loops, spreading it across several functions and files, and using global variables or different data types.

Paraphrasing Boland and Black [4], many test suites are structured so that the test cases can be analyzed or compiled individually or as a single, large program. Figure 1 depicts the quantity, size, and origin of cases in each language. More details of the content, organization, and use of SARD are in Ref. [5].

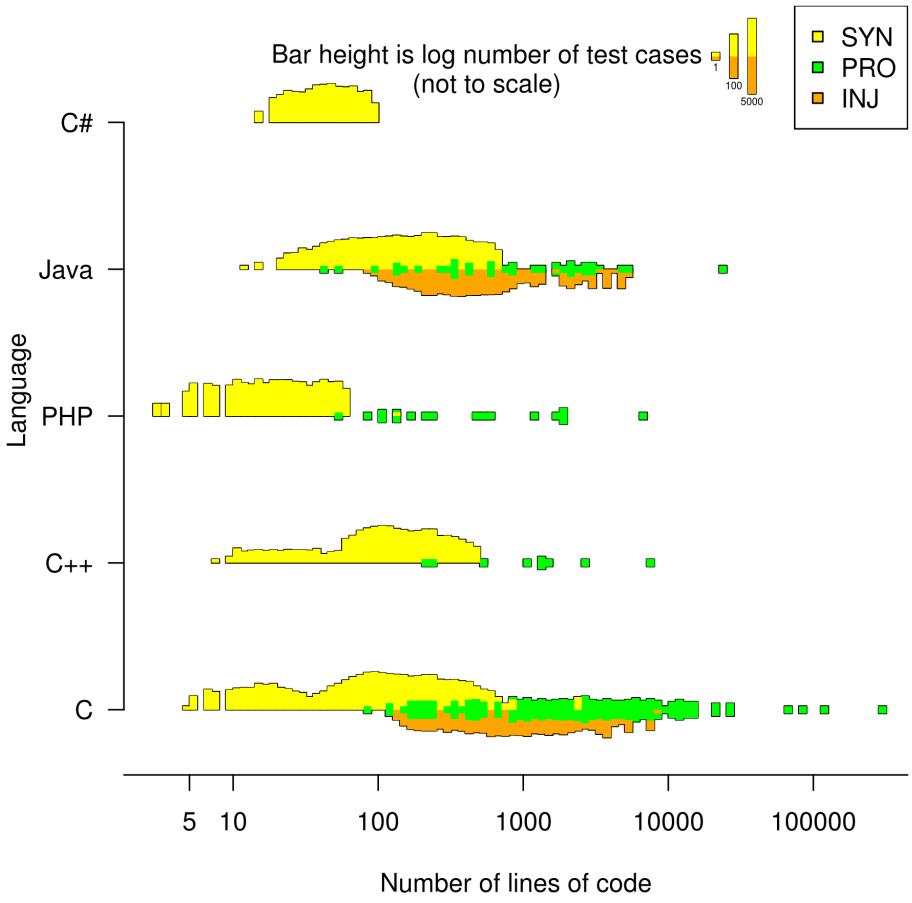

Fig. 1. Number of test cases in SARD by language and size as of June 2017. The X axis is the number of lines of code plotted on a logarithmic scale. The height of each bar is the logarithm of the number of test cases with that many lines of code. Synthetic cases (SYN), in which all weaknesses are known, are yellow. Production code cases (PRO), which have some weaknesses identified, are green. Cases with weaknesses injected (INJ) into production code are orange.

3. Sources and the Future

Since it is not clear what the ultimate test suite would be (or if there is one!), we gather test cases from many sources, as detailed in Ref. [5]. For example, synthetic cases came from Lincoln Laboratory, the U.S.

National Security Agency's Center for Assured Software (CAS), and students at TELECOM Nancy. Other sources of SARD test cases are companies donating code used for regression testing, research and academic efforts giving the suites they developed, and government programs contributing thousands of cases. One program derived thousands of cases by injecting bugs into a dozen base applications. Authors of software assurance books contributed their examples, too. Representing operational code, SARD includes code used in the five Static Analysis Tool Expositions [6] (e.g., WordPress, Apache Tomcat, Wireshark, and Chrome), as well as slices of code taken from popular Internet applications (e.g., BIND and Sendmail).

SARD is archival: once a case is added, it will not be changed or deleted. If a case has problems, it may be marked as *deprecated* and a replacement added.

We frequently add new collections. We plan to add cases in other languages, such as Solidity, JavaScript, Python, Haskell, or Lua. SARD can host system designs and compiled binaries, in addition to source code.

4. Impact

Like spell checkers, software assurance tools can report some problems before they reach production. Developers can learn about the strengths and limitations of a candidate tool by running it on programs with known bugs, like those in SARD. Since most bugs are recorded in metadata, results can be evaluated semi-automatically, indicating the kinds of bugs that a tool finds (and does not find) and a false positive rate.

Security analysts, users, and tool developers have cut months off the time needed to evaluate a tool or technique using SARD test cases. Educators refer students to SARD for examples of weaknesses. Having a reliable and growing body of code with known weaknesses helps the community improve software assurance tools themselves and encourage their appropriate use. We invite software researchers and project managers to donate their test suites and cases.

Acknowledgments

The author thanks David Flater and Gabriel Sarmanho for help with the chart in Fig. 1 and thanks John Henry Scott for valuable suggestions.

5. References

[1] Software assurance reference dataset (SARD). https://samate.nist.gov/SARD/. Accessed 4 January 2018.
[2] Common weakness enumeration. https://cwe.mitre.org/. Accessed 4 January 2018.
[3] Bojanova I, Black PE, Yesha Y, Wu Y (2016) The bugs framework (BF): A structured approach to express bugs. *2016 IEEE International Conference on Software Quality, Reliability, and Security (QRS)*, pp 175–182. https://doi.org/10.1109/QRS.2016.29.
[4] Boland T, Black PE (2012) Juliet 1.1 C/C++ and Java test suite. *IEEE Computer* 45(10):88–90. https://doi.org/10.1109/MC.2012.345.
[5] Black PE (2017) SARD: Thousands of reference programs for software assurance. *Journal of Cyber Security and Information Systems - Tools & Testing Techniques for Assured Software - DoD Software Assurance Community of Practice: Volume 2* 5(3):6–13.
[6] Cohen TS, Cupif D, Delaitre A, De Oliveira CD, Fong E, Okun V (2017) Improving software assurance through static analysis tool expositions. *Journal of Cyber Security and Information Systems - Tools & Testing Techniques for Assured Software - DoD Software Assurance Community of Practice: Volume 2* 5(3):14–22.

About the author: *Paul E. Black has been with NIST since 1997. He has nearly 20 years of industrial experience in areas such as developing software for microelectronic design and verification, assuring software quality, and managing business data processing. He edits the on-line Dictionary of Algorithms and Data Structures (DADS) and started the Software Assurance Metrics And Tool Evaluation (SAMATE) team. The National Institute of Standards and Technology is an agency of the U.S. Department of Commerce.*