ClaBureDB: Classified Bug-Reports Database

Tool for developers of program analysis tools

Jiri Slaby, Jan Strejček, and Marek Trtík

Faculty of Informatics, Masaryk University
Botanická 68a, 60200 Brno, Czech Republic
{slaby,strejcek,trtik}@fi.muni.cz

Abstract. We present a database that can serve as a tool for tuning and evaluation of miscellaneous program analysis tools. The database contains bug-reports produced by various tools applied to various source codes. The bug-reports are classified as either real errors or false positives. The database currently contains more than 800 bug-reports detected in the Linux kernel 2.6.28. Support of other software projects written in various programming languages is planned. The database can be downloaded and manipulated by SQL queries, or accessed via a web frontend.

1 Introduction

Many successful bug-finding tools based on various program analysis methods appeared during the last ten years. None of them is perfect. Each tool either reports both real errors and false positives, or it discovers only a part of real errors. To improve or evaluate such a tool, one needs to run the tool on some source codes and then analyze the obtained bug-reports[1], i.e. classify them as false positives or real errors, and find errors in the sources that were missed by the tool. This work is usually tedious and time consuming, especially when one tunes or studies performance of a tool for real software projects. The tedious work can be avoided if suitable benchmarks, i.e. programs with information about their errors, are available.

There exist benchmark suites consisting of small synthetic programs [1, 2, 4, 6, 7] and those consisting of real-world programs [2, 3, 5]. In benchmark suites [1, 4, 6], relevant program locations in small synthetic programs are explicitly marked as either erroneous or safe. The benchmark suite [2] marks only known real errors. The situation is different for benchmarks with real-world programs: [5] contains marked test cases triggering/non-triggering errors, while [3] provides bug-reports (both real errors and false positives mixed together) produced by Findbugs and sorted according to priority levels assigned to the reports by the tool. The benchmark suites discussed so far consider different error types and

[1] We deliberately use term "bug" in the paper. It stems from the need of the database to comprehend for example coding style violations. In those cases, commonly used terms like "failure" or "fault" fail to apply.

provide their own error taxonomies with exception of suites [1, 2], where errors types are linked to *Common Weakness Enumeration* (CWE) [10].

As far as we know, there is no benchmark suite containing big real-life projects with a significant list of uniformly described bug-reports classified as real errors or false positives. Our benchmark suite ClaburéDB should fill this gap.

ClaburéDB currently contains only a single project, namely the Linux kernel 2.6.28. We have collected about 850 bug-reports of 11 error types produced by several bug-detection tools run on the kernel. The reports have been manually classified as either real errors or false positives by skilled programmers with a help of Linux kernel developers. In fact, it would be sufficient to store only real errors assuming that we know all of them (and thus we can assume that all other bug-reports are false positives). As this assumption is completely unrealistic for a real-life project, we store both real errors and false positives.

The database is still developing in several directions: we plan to add more bug-reports for the Linux kernel, to support other software projects, and to augment the web interface to allow other users to add and maintain the database content.

The database can be downloaded in the SQLite 3 format for local use under the *Open Database License v1.0* [11] or accessed via a web interface at:

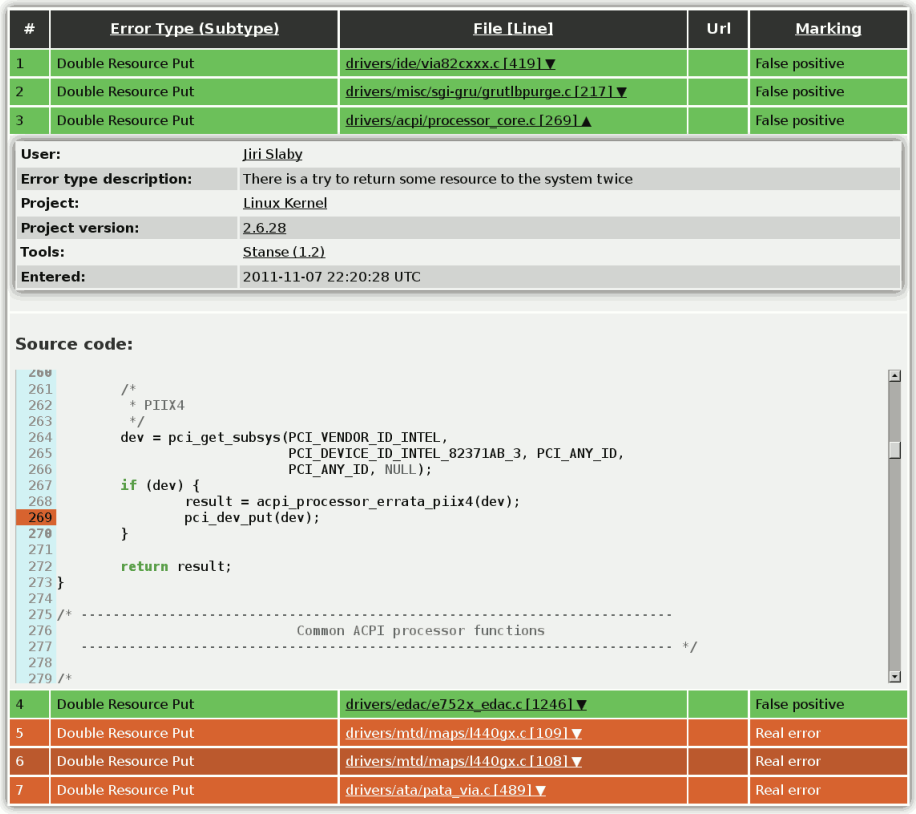
`http://claburedb.fi.muni.cz/`

The paper is organized as follows. The following section introduces the basic structure of the database and our web interface. Section 3 describes the current content of the database including considered kinds of errors and an overview of collected bug-reports and their sources. Section 4 suggests possible use of the database for evaluation of a program analysis tool. Finally, the last section presents our future plans with ClaburéDB.

2 Database Structure

The database is designed to accommodate various kinds of errors from diverse projects and project versions. As projects can be written in arbitrary programming languages, can contain very specific kinds of errors, can be maintained by different teams, and can be interesting for distinct user groups, we decided to have each project in a separate sub-database. A sub-database comprises information about considered error types, bug-reports, users, tools, and relations between them. There are two main tables in each sub-database:

error_type This table keeps a specification of considered kinds of errors. We will outline some of possible types later in Section 3.2. The table contains a name of the type, short description and a reference CWE number if exists (see later).



#	Error Type (Subtype)	File [Line]	Url	Marking
1	Double Resource Put	drivers/ide/via82cxxx.c [419]▼		False positive
2	Double Resource Put	drivers/misc/sgi-gru/grutlbpurge.c [217]▼		False positive
3	Double Resource Put	drivers/acpi/processor_core.c [269]▲		False positive

User:	Jiri Slaby
Error type description:	There is a try to return some resource to the system twice
Project:	Linux Kernel
Project version:	2.6.28
Tools:	Stanse (1.2)
Entered:	2011-11-07 22:20:28 UTC

Source code:

```
261         /*
262          * PIIX4
263          */
264         dev = pci_get_subsys(PCI_VENDOR_ID_INTEL,
265                              PCI_DEVICE_ID_INTEL_82371AB_3, PCI_ANY_ID,
266                              PCI_ANY_ID, NULL);
267         if (dev) {
268                 result = acpi_processor_errata_piix4(dev);
269                 pci_dev_put(dev);
270         }
271
272         return result;
273 }
274
275 /* --------------------------------------------------------------------------
276                               Common ACPI processor functions
277    -------------------------------------------------------------------------- */
278
279 /*
```

4	Double Resource Put	drivers/edac/e752x_edac.c [1246]▼		False positive
5	Double Resource Put	drivers/mtd/maps/l440gx.c [109]▼		Real error
6	Double Resource Put	drivers/mtd/maps/l440gx.c [108]▼		Real error
7	Double Resource Put	drivers/ata/pata_via.c [489]▼		Real error

Fig. 1. List of seven bug-reports with one record expanded showing detailed information about the bug-report.

error Each line in this table corresponds to one bug-report. It is specified by the error type, location (usually file and line), URL for reference (to provide more information), classification (false positive, real error, unclassified), confirmation (an argument supporting the validity of the bug-report classification, e.g. a commit ID of the corresponding bug-fix), user who inserted the entry, and timestamp of the moment of insertion.

Figure 1 depicts a list of seven bug-reports of type *Double Resource Put* as presented in the ClabureDB's web interface at `http://claburedb.fi.muni.cz/`. Four of these reports are false positives (green), the other three are real errors (orange). One of the reports is expanded, so that we can see its details and a highlighted source code with a marked error line.

Recall that the database can be also downloaded in the SQLite 3 format for local use under the *Open Database License v1.0* [11].

3 Current Contents of the Database

ClaburеDB currently contains only bug-reports produced for the Linux kernel 2.6.28. We have chosen Linux kernel for several reasons: it is a big (20484 files with nearly 9 millions LOC in total), real-life, self-contained, and open-source project with several public sources of information about errors. Moreover, the kernel contains many distinct parts (core, hardware drivers, filesystems, networking etc.) that cover some standard application areas of program analysis tools.

3.1 Sources of Bug-Reports

We used three program analysis tools to gather bug-reports for the kernel: Clang 3 [9], Stanse 1.2 [8], and one commercial tool which wanted to stay in anonymity. Note that Clang does not natively detect locking errors in the Linux kernel. Hence, we slightly modified `experimental.unix.PthreadLock` checker just to understand kernel locking functions. The database also comprises bug-reports from kernel and Novell bug tracking systems and mailing lists. For that purpose we implemented our own web crawler. Finally, there are few bug-reports detected manually.

3.2 Linux Kernel Error Types

We have analyzed output of available tools applicable to the Linux kernel and we have decided to focus on the following 11 error types. Nine of these error types are present in *Common Weakness Enumeration* (CWE) [10] and we provide their CWE identification numbers. The two remaining error types are specific for the Linux kernel (they can be seen as a violation of the kernel coding policy) and thus they are not covered by CWE.

The list of considered error types follows. For each error type we explicitly describe the program location associated with an error of this type. This is to evade misinterpretation, because diverse tools can associate the same error with different program locations. For example, an error present at an outgoing edge of a function may be associated to the opening brace of the function, the closing brace, or to the corresponding `return` statement.

BUG/WARNING (CWE 617) Developers often inject *assert*s to their code, e.g. to ensure that their function is given a correct input. For example, a destroy function of an object `obj` can contain a line `assert(!obj->active)` to check that the object to be destroyed is inactive. An error occurs if a condition of some `assert` is violated.
Error location: the line with the violated assertion.

Division by Zero (CWE 369) The code contains a division instruction but the actual value of the divisor is zero.
Error location: the line with the division.

Double Lock (CWE 764) Some lock is locked by a thread twice in a row and it leads to an inconsistent lock state. Note that we ignore double locks of semaphores as this may be their intentional application.
Error location: the line with the second call of lock.

Double Unlock (CWE 765) Some lock is unlocked by a thread twice in a row and it leads to an inconsistent lock state. Again, we ignore double unlocks of semaphores.
Error location: the line with the second call of unlock.

Double Free (CWE 415) A freeing function is called twice on the same address while no reassignment to the passed pointer occurred between the two calls. Most allocators can detect this and usually kill the program.
Error location: the line of the invalid (second) free.

Memory Leak (CWE 401) Some code omits to free a memory which was allocated previously.
Error location: the line with the allocation statement.

Invalid Pointer Dereference (CWE 465) The code tries to access some memory, but the pointer used is invalid. It may become invalid in many ways. For example, the pointer may point to a released memory (known as *dangling pointer* or *use after free*) or it can be set to NULL (known as *NULL pointer dereference*) or uninitialized. Another source of the problem may be accessing an array out of bounds.
Error location: the line of the dereference.

Double Resource Put (CWE 763) The code requests one copy of a resource (e.g. a structure holding hardware status) from a system, but there is more than one attempt to put that resource back to the system. Like in this example:

```
struct pci_dev *pdev = pci_get_device(vendor, device, NULL); // get
if (pdev) {
  work_with_pdev(pdev)
  pci_dev_put(pdev);                // first put
}
pci_dev_put(pdev);                  // second (illegal) put of the same
```

Error location: the line of the second put.

Resource Leak (CWE 404) The code requests a resource from a system, but omits to return that back. For example, the error occurs in the previous example if we remove both `pci_dev_put` calls.
Error location: the line of the request/get.

Calling Function from Invalid Context (not in CWE, Linux kernel specific) Some function is called at an inappropriate place or within an invalid context. This includes calling functions like sleep or wait inside spin-lock critical sections or in interrupt handlers. This is considered to be an error because results of such a call are undefined: the system may become unresponsive or may crash for instance.
Error location: the line of the inappropriate call.

Leaving Function in Locked State (not in CWE, Linux kernel specific) This error type originates from the kernel requirements that a process has to release all locks before returning control to userspace. This error occurs if

Error Type	Real Err.	False Pos.	Unclassified	Total
BUG/WARNING	8	0	0	8
Division by Zero	2	0	0	2
Double Lock	16	95	4	115
Double Unlock	22	90	9	121
Double Free	0	1	0	1
Memory Leak	7	13	0	20
Invalid Pointer Dereference	17	17	0	34
NULL Pointer Dereference	17	14	0	31
Use After Free	0	3	0	3
Double Resource Put	3	4	0	7
Resource Leak	13	51	24	88
Calling function from invalid context	16	19	0	35
Leaving function in locked state	30	352	37	419
Overall Count	134	642	74	850

Table 1. Reports in ClabureDB.

a function has an execution path where some lock is locked and left locked when leaving the function. It is considered to be an error (violation of the kernel coding policy) as such an execution may lead to a deadlock on the next invocation of any function wanting to take the same lock.
Error location: the line of the corresponding `return` statement or closing brace if there is no `return`.

3.3 Bug-Reports in the Database

Currently the database comprises 850 reports: 134 real errors, 642 false positives, and 74 entries which are unclassified. Many of the unclassified entries were added even recently and are about to be classified soon. Table 1 depicts how each kind of bug-reports is represented in the database. As seen from the table, most of the bug-reports in the database are related to locking errors.

4 Intended Use of the Database

Typical use of ClabureDB is tuning and evaluation of a bug-finding program analysis tool. To evaluate such a tool, one chooses a project (or its part) from our benchmark suite and run the tool on these source codes. As the second step, the sub-database of classified bug-reports corresponding to the chosen project is downloaded from ClabureDB. The installation and local database usage is described in the ClabureDB documentation.

The bug-reports produced by the tool are compared to the downloaded data. This way, one immediately gets a classification (real error/false positive) of all the reports matched in the database and also information about missed real errors if

there are any. It remains to manually classify the bug-reports unmatched with the database content. The numbers of produced false positives, detected and missed real errors can be further statistically processed according to methodologies of [2, 3] which produce several metrics including accuracy and precision. The missed errors and false positives are valuable inputs for tuning the tool.

Finally, the user is kindly asked to insert newly discovered bug-reports with their classification into the database.

5 Conclusion and Future Plans

We have presented ClabureDB: the database of classified bug-reports. It is intended as an open platform for sharing valuable benchmarks for developers of program analysis tools. The database already contains a large collection of uniformly described bug-reports produced by various tools on a large real-life project, namely the Linux kernel 2.6.28.

We plan to extend the database in several directions. Namely, we intend to collect more bug-reports and to support more error types in the Linux kernel project. For example, we plan to add error types connected to deadlock and livelock. Further, we plan to add other real-life projects. We also plan to extend the web interface to enable developers of program analysis tools to maintain and augment the database.

In general, the future of ClabureDB depends on a feedback from the program analysis community. At this point, we would like to encourage you to contact us at `claburedb@fi.muni.cz` if you have any comments, suggestions (for example which projects should be added to the database), questions, or sources of bug-reports for the Linux kernel or other interesting projects. We would especially welcome people willing to participate on creating and maintaining the database content for another project.

Acknowledgements All authors have been supported by The Czech Science Foundation (GAČR), grant No. P202/12/G061.

References

1. G. Chatzieleftheriou and P. Katsaros. Test-driving static analysis tools in search of C code vulnerabilities. In *Proceedings of COMPSACW*, pages 96–103. IEEE Computer Society, 2011.
2. C. Cifuentes, Ch. Hoermann, N. Keynes, S. Long L. Li, E. Mealy, M. Mounteney, and B. Scholz. BegBunch – Benchmarking for C Bug Detection Tools. In *Proceedings of DEFECTS*, pages 16–20. ACM, 2009.
3. S. Heckman and L. Williams. On establishing a benchmark for evaluating static analysis alert prioritization and classification techniques. In *Proceedings of ESEM*, pages 41–50. ACM, 2008.
4. K. Kratkiewicz. Using a diagnostic corpus of C programs to evaluate buffer overflow detection by static analysis tools. In *Proceedings of BUGS*, 2005.

5. S. Lu, Z. Li, F. Qin, L. Tan, P. Zhou, and Y. Zhou. Bugbench: Benchmarks for evaluating bug detection tools. In *In Workshop on ESDDT*, 2005.
6. T. Newsham and B. Chess. ABM: A prototype for benchmarking source code analyzers. In *Proceedings of SSATTM*, pages 52–59. NIST Special Publication, 2005.
7. NIST. Samate reference dataset project. `http://samate.nist.gov/SRD/`.
8. J. Obdržálek, J. Slabý, and M. Trtík. STANSE: Bug-finding Framework for C Programs. In *Proceeding of MEMICS*, LNCS, pages 167–178, 2011.
9. Clang: a C language family frontend for LLVM. `http://clang.llvm.org/`.
10. Common Weakness Enumeration (CWE). `http://cwe.mitre.org/`.
11. Open Database License 1.0. `http://opendatacommons.org/licenses/odbl/1.0/`.