CBCD: Cloned Buggy Code Detector

Technical Report
UW-CSE-11-05-02

May 2, 2011
(Revised March 20, 2012)

Jingyue Li
DNV Research&Innovation
Høvik, Norway
Jingyue.Li@dnv.com

Michael D. Ernst
U. of Washington
Seattle, WA, USA
mernst@uw.edu

CBCD: Cloned Buggy Code Detector

Jingyue Li
DNV Research & Innovation
Høvik, Norway
Jingyue.Li@dnv.com

Michael D. Ernst
University of Washington
Seattle, WA, USA
mernst@uw.edu

***Abstract*—Developers often copy, or clone, code in order to reuse or modify functionality. When they do so, they also clone any bugs in the original code. Or, different developers may independently make the same mistake. As one example of a bug, multiple products in a product line may use a component in a similar wrong way. This paper makes two contributions. First, it presents an empirical study of cloned buggy code. In a large industrial product line, about 4% of the bugs are duplicated across more than one product or file. In three open source projects (the Linux kernel, the Git version control system, and the PostgreSQL database) we found 282, 33, and 33 duplicated bugs, respectively. Second, this paper presents a tool, CBCD, that searches for code that is semantically identical to given buggy code. CBCD tests graph isomorphism over the Program Dependency Graph (PDG) representation and uses four optimizations. We evaluated CBCD by searching for known clones of buggy code segments in the three projects and compared the results with text-based, token-based, and AST-based code clone detectors, namely Simian, CCFinder, Deckard, and CloneDR. The evaluation shows that CBCD is fast when searching for possible clones of the buggy code in a large system, and it is more precise for this purpose than the other code clone detectors.**

***Keywords*- Validation, Debugging aids**

I. INTRODUCTION

Although copy-paste is generally regarded as a bad coding practice, it is sometimes necessary, and some developers do it to save development effort. Baker found that 24% of files examined included exact matches of code lines [4]. Ducasse et al. reported that two files of gcc have more than 60% duplication [3]. A study of code clones in Linux [2] showed that:

- A few copy-pasted segments were copied more than eight times.
- Device drivers and cryptography have the highest percentage of clones, because many drivers share similar functionality and cryptographic algorithms consist of multiple similar computational steps.

Code copy-paste and software reuse makes buggy code appear in multiple places in a system or in different systems. For example, code clones and software reuse have caused duplicated software security vulnerabilities [18]. Cut-and-paste is a major cause of operating system bugs [11].

This paper makes two contributions. First, we examined the data in the SCM (Software Configuration Management System) of 4 projects: an industrial software product line, the Linux kernel, Git, and PostgreSQL. We discovered that identical buggy code does exist in all 4 projects.

Second, to find clones of buggy code, we developed a clone detection tool, CBCD. Given an example of buggy code, CBCD uses isomorphism matching in the Program Dependence Graph (PDG) [15] to search for identical code — that is, clones. Subgraph isomorphism is NP-complete [13], so we implemented four optimizations that reduce the number and complexity of graphs in the PDG isomorphism matching. Evaluation of CBCD on real cloned buggy code confirms that CBCD is scalable to large systems. To evaluate how well CBCD can find cloned bugs, we also compared CBCD with text-based, token-based, and AST-based code clone detectors, using the identified buggy codes and their clones as oracles. CBCD outperformed the other approaches. (Our evaluation focuses on the important problem of finding clones of buggy code. For other tasks, the other clone detectors may be better than CBCD.)

The rest of this paper is organized as follows. Section 2 presents our empirical study of cloned buggy code in one commercial product line and three large open source systems. Section 3 describes the design and implementation of CBCD, which can find cloned buggy code. Section 4 presents our experimental evaluation. Section 5 discusses related work, and Section 6 concludes.

II. AN EMPIRICAL STUDY OF CLONED BUGGY CODE

We first manually investigated whether buggy lines of code are cloned in real systems. We examined the SCM of the Linux kernel, Git, and PostgreSQL, and the bug reporting system of a commercial software product line.

A. The Linux Kernel

For the Linux kernel, we searched for the keywords in Table I in commit messages and in the bug tracking system, which records discussions between developers during debugging. For each match, we read the description of the commit, the discussions between developers, and the "diff" of the original file and the changed file. This information indicated to us whether the commit was necessitated by duplication of a bug. If so, we identified the buggy code and its clones manually.

The second column of Table I shows the number of distinct, independent bugs that exist in multiple locations. By distinct, we mean that we count a bug once, even if it appears in 3 places. By independent, we mean that if a commit message said, "The same problem as commit #1234", we count only one of the two bugs. Finally, there is no double-

counting: if a commit message said "the same problem as #1234, with the same fix", then it only appears in one row of Table I. Some examples of these cloned bugs are shown in Table II. However, for some of these bugs, we cannot locate the cloned buggy code, because the developers did not give enough details. The third column of Table I omits such bugs. For example, one developer said, "The same bug that existed in the 64bit memcpy() also exists here so fix it here too" but did not specify which version of which file of the system includes the fix of the bug in 64bit memcpy(). As there are many files and many versions of Linux, it would be difficult to search all of them to find the fixes to memcpy(). Even if we found a change to memcpy(), without further information, we do not know if that change is the fix mentioned by the developer.

TABLE I. CLONED BUGS WHICH EXIST IN MORE THAN ONE PLACE IN THE LINUX KERNEL

Key words used for searching the SCM	Number of distinct bugs existing in more than one place	Number of bugs whose clones we can locate
same bug	53	23
same fix	48	24
same issue	62	39
same error	7	6
same problem	112	65
Sum	282	157

TABLE II. EXAMPLES OF CLONED BUGS IN THE LINUX KERNEL

Phrases in the SCM explaining the cloned bugs	Code modified (i.e., the lines of code modified by the bug fix)
This is quite the same fix as in 2cb96f86628d6e97fcbda5fe4d8d74876239834c	static int my_atoi(const char *name){ int val = 0; for (;; name++) { switch (*name) { case '0' ... '9': val = 10*val+(*name-'0'); break; default: return val;} }}
This patch fixes iwl3945 deadlock during suspend by moving notify_mac out of iwl3945 mutex. This is a portion of the same fix for iwlwifi by Tomas.	ieee80211_notify_mac(priv->hw, IEEE80211_NOTIFY_RE_ASSOC);
It turns out that at least one of the caller had the same bug.	ret = btrfs_drop_extents(trans, root, inode, start, aligned_end, start, &hint_byte);
Other platforms have this same bug, in one form or another	atomic_inc(&call_data->finished); func(info);

B. Git and PostgreSQL

For the Git and PostgreSQL projects, we used the same methodology. Table III shows the number of bugs that exist in multiple places.

C. A Commercial Software Product Line

We also evaluated a commercial product line in which a single product is produced for more than 40 different operating systems and mobile devices. For 17 of the projects, we have access to bug reports and developer discussions. These projects have a total of 25420 valid bugs that are confirmed and resolved as a bug in the code, not a user error.

We searched for the same keywords in the bug reports. Unlike the Linux kernel, Git, and PostgreSQL, we do not have full access to the source code in the SCM. Thus, we did not check the code differences. Our assessment of whether a bug was duplicated (as shown in Table IV) was based on reading the discussions between developers during debugging. It turns out that 3.8% (969/25420) of the bugs in these 17 projects exist in more than one place.

TABLE III. CLONED BUGS WHICH EXIST IN MORE THAN ONE PLACE IN GIT AND POSTGRESQL

Key words used for searching the SCM	GIT		POSTGRESQL	
	Number of distinct bugs existing in more than one place	*Number of bugs whose clones we can locate*	*Number of distinct bugs existing in more than one place*	*Number of bugs whose clones we can locate*
same bug	7	5	9	9
same fix	7	4	5	4
same issue	14	3	2	0
same error	0	0	1	8
same problem	5	0	16	1
Sum	33	12	33	22

TABLE IV. CLONED BUGS WHICH EXIST IN MORE THAN ONE PLACE IN THE COMMERCIAL SOFTWARE PRODUCT LINE

Key words used for searching the bug reports	Number of distinct bugs existing in more than one place
same bug	170
same fix	40
same issue	302
same error	56
same problem	401
Sum	969

III. CBCD, A TOOL TO SEARCH FOR CLONED BUGGY CODE

Once a bug is detected, it is necessary to check the whole system to see if the bug exists somewhere else. Section II shows that this is not merely a theoretical concern, but is important in practice. It is especially important for a software product line, because of high similarity among products. Customer satisfaction drops when a customer re-encounters a bug that the vendor claimed to have fixed. Although regression testing can check whether a bug is fixed, or can detect an identical manifestation of the bug in other products, regression testing cannot find all occurrences of the bug, especially when testers do not know where the buggy code may appear. Thus, it is important to supplement regression testing by a search for clones to locate code that may behave similarly to the buggy code.

A. PDG Based Code Clone Detectors

Some buggy lines may be copy-pasted "as-is", but often, developers slightly modify the copy-pasted code to fit a new context [2]. More than 65% of copy-pasted segments in Linux require renaming at least one identifier, and code insertion and deletion happened in more than 23% of the

copy-pasted segments [2]. Statement reordering, identifier renaming, and statement insertion or deletion are also common in buggy code clones, especially clones introduced due to code or component reuse. For example, in Table II, a developer stated that "Other platforms have this same bug, in one form or another."

Our approach is to adapt Program Dependence Graph (PDG)-based code clone detection methods [7, 8, 9, 10], because we believe that the PDG-based approach is more resilient to code changes than text-based, token-based, and AST-based approaches.

B. *Tool Architecture*

Our tool, CBCD (for "Cloned Buggy Code Detector") has a pipe-and-filter architecture, as shown in Fig. 1. CBCD represents a program or code fragment as a PDG, which is a directed graph. Each vertex represents an entity of the code, such as a variable, statement, and so on; CBCD also records the vertex kind (e.g., "control-point", "declaration", or "expression"), the position (i.e., the file name and the line of the represented source code), and the source code text itself. Each edge of a PDG represents control or data dependency between two vertexes.

CBCD's algorithm consists of three steps.

Step 1: CodeSurfer [14] generates the PDG of both the buggy code (the "Bug PDG") and of the system to be searched for clones of the buggy code (the "System PDG"). The Bug PDG may consist of multiple sub-graphs depending on the structure of the buggy code; CBCD handles this case, but for simplicity of presentation this paper assumes the Bug PDG is connected. The System PDG consists of a collection of interlinked per-procedure PDGs.

Step 2: CBCD prunes and splits the System PDG (see Section III.C) to reduce its complexity and make subgraph checking cheaper. Optionally, CBCD also splits the original Bug PDG into multiple smaller PDGs (see Section III.C.4).

Step 3: CBCD determines whether the Bug PDG is a subgraph of the System PDG. It uses igraph's [16] implementation of subgraph isomorphism matching. igraph is faster than other tools, such as Nauty [17], when comparing randomly-connected graphs with less than 200 nodes [12].

CBCD filters the matches reported by igraph. CBCD only outputs matches where, for each corresponding vertex, the vertex kinds match and the source code text matches. When comparing vertex kinds, CBCD tolerates control replacement, e.g., when developers change a "for" loop to a "while" loop to provide the same functionality. When comparing source code text, vertexes that represent parameters of a function call are exempted. Note that even if all vertex kinds and text match identically (which CBCD does not require), the source code could still be different so long as it led to the same PDG. For example, reordering of (non-dependent) statements does not affect the PDG, nor does insertion of extra statements, such as debugging printf statements.

CBCD aims to find all semantically identical code clones. Two code snippets are semantically identical if there is no program context that can distinguish them—that is, if one snippet is substituted for the other in a program, the program behaves identically to before, for all inputs. Determining semantic equivalence is undecidable, so CBCD reports code with matching PDGs. As a result, every match that CBCD finds is semantically identical to the buggy code, but CBCD is not guaranteed to find all semantically-identical clones.

C. *Pruning the Search Space for Isomorphism Graph Matching*

All code clone detection tools that rely on graph matching face scalability problems. CBCD's isomorphism matching step is the most time-consuming step, especially for matching two big graphs. The reason for this is that subgraph isomorphism identification is NP-complete [13]. In the worst case, the fast subgraph isomorphism algorithm [12] implemented by igraph [16] requires O(N!N) time, where N is the sum of the number of nodes and edges of both graphs to be compared. Liu et al. [9] claim that "PDGs cannot be arbitrarily large as procedures are designed to be of reasonable size for developers to manage." In practice, a procedure can be very big. For example, we used Git as a subject program, and its "*handle_revision_opt*" procedure has 817 vertexes and 2479 edges. But, even smaller comparisons can be intractable in practice. Consider a modest example: the buggy code has 5 lines of code (with around 10 vertexes and 15 edges in the PDG) and the procedure has 100 lines of code (around 200 vertexes and 300 edges). In this example, N = 525 and N!N is 3.6×10^{1204}.

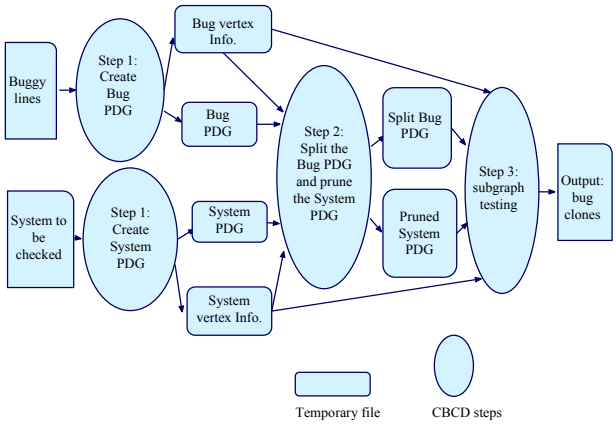

Figure 1. Architecture of CBCD

To deal with the scalability problem, Step 2 of CBCD prunes the number and complexity of the graphs to be compared.

We have implemented four optimizations. The first three optimizations are *sound*: each never excludes a true match, but makes the algorithm faster overall. These optimizations are run by default. The fourth optimization runs only if the buggy code segment contains too many lines of code.

The first three optimizations are based on the fact, explained in Section III.B, that CBCD reports system code as a clone of buggy code only if both the shape of the respective PDGs, and also the vertex kind and source text of corresponding vertices, are identical. The first three optimizations can be viewed as enhancements to the

subgraph isomorphism checker, working around its limitation that it does not account for vertex kinds and source text.

All four optimizations are also based on the following observation: In most cases, the Bug PDG is small. Fig. 2 validates this observation: it is the maximum number of contiguous lines of code in each of the 163 Git, Linux kernel, and PostgreSQL bugs for which we can locate their cloned bugs. (This excludes 28 bug fixes that added code rather than changing code.) More than 88% of the bugs cover 4 or fewer contiguous lines of code.

1) Optimization 1 (Opt1): Exclude Irrelevant Edges and Nodes from the System PDG

CBCD removes every edge that cannot match an edge in the Bug PDG, because such an edge is irrelevant for CBCD's purposes. In particular, CBCD removes every edge whose start and end vertex kinds and vertex text are not included in the start and end vertex kinds and characters of an edge in the Bug PDG. In the best case, this disconnects entire sets of nodes, but it is useful even if it merely removes edges, because a single System PDG can be very big.

For example, suppose the Bug PDG has two edges: one from vertex kind "control-point" to vertex kind "expression", and the other from "expression" to "actual-in". Then, CBCD excludes from the System PDG all edges that do not start with "control-point" and end with "expression", or start with "expression" and end with "actual-in".

At this point, CBCD also compares the vertex characters (source code text), for vertex kinds whose code must match (e.g., not procedure parameters nor arguments). CBCD discards those with text that cannot match the Bug PDG. The purpose of comparing vertex kinds and characters is different than Step 3 of Section III.B. The comparison here excludes System PDG vertexes and edges that are irrelevant to the Bug PDG. The comparison in Step 3 ensures that the vertexes in the isomorphism matching graphs are also identical.

2) Optimization 2 (Opt2): Break the System PDG into Small Graphs

This optimization transforms the System PDG from one large graph into multiple small ones. CBCD must run more subgraph isomorphism matchings, but each matching will focus on a smaller graph. The idea is to utilize the vertex kind information of the Bug PDG to choose only small sections of the procedure PDG for each subgraph isomorphism matching. The steps of Opt2 are:

- **Opt2-step1:** Count the number of nodes of each vertex kind in the Bug PDG and the System PDG.
- **Opt2-step2:** Choose the vertex kind vk_{min} in the Bug PDG that has the minimum number of occurrences in the System PDG. If it occurs 0 times in the System PDG, there is no graph match.
- **Opt2-step3:** Calculate the pseudo-radius d_b of the Bug PDG: the greatest distance between a node of vertex kind vk_{min} and any other node.
- **Opt2-step4:** For each node of vertex kind vk_{min} in the System PDG, find the neighbor graph of the vertex, with radius d_b from the node of kind vk_{min}. The distance computations ignore edge directions.

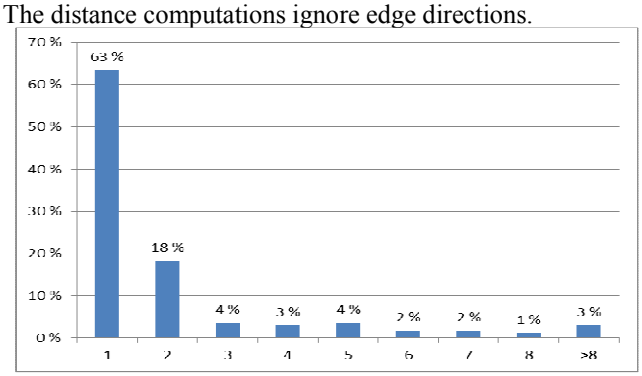

Figure 2. Size (contiguous lines) of the largest component of each bug fix

Fig. 3 shows an example. Since the nodes of vertex kind vk_{min} must match, and there are few of them, it makes sense to check subgraph isomorphism only near them. It is possible for the neighbor graphs to overlap, in which case some PDG nodes appear in multiple distinct neighbor graphs and will be tested for isomorphism with the Bug PDG multiple times.

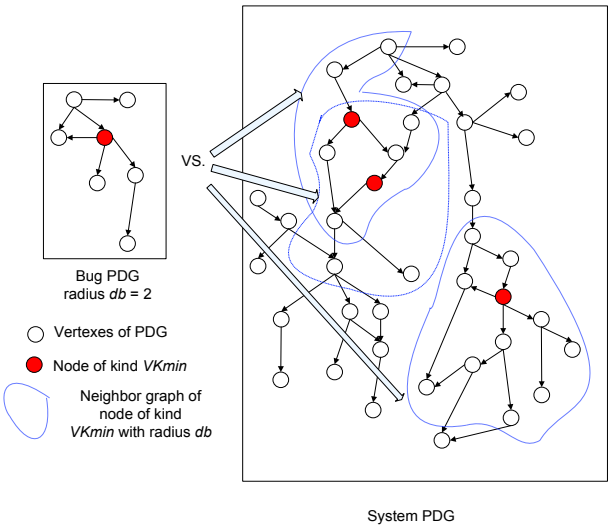

Figure 3. Breaking the System PDG into smaller pieces (Opt2)

Opt2 adds some extra overhead to CBCD. Here is the theoretical analysis of the time complexity without Opt2 and with Opt2. We assume that the Bug PDG has i_1 nodes and j_1 edges and the System PDG has i_2 nodes and j_2 edges. Then the time complexity of each step of Opt2 is:

- Opt2-step1. $O(i_1+i_2)$
- Opt2-step2. $O(1)$
- Opt2-step3. $O(i_1\ j_1)$, because of the igraph_diameter() function of igraph [16].
- Opt2-step4: $O(w(i_2+j_2))$, where there are w vertexes in the System PDG having the chosen vertex kind from Opt2-step2, because of igraph_neighborhood_graph() function of igraph [16] .

Although Opt2 adds the above overhead, it can significantly reduce the time complexity of Step 3 of Section III.B, i.e. subgraph isomorphism matching.

Without Opt2, the time complexity of comparing the Bug PDG and the System PDG is between $O((i_1+ j_1+ i_2+ j_2)^2)$ and $O((i_1+ j_1+ i_2+ j_2)!\ (i_1+ j_1+ i_2+ j_2))$, for the algorithm [12] implemented by igraph.

Since each subgraph of the System PDG has identical pseudo-radius as the Bug PDG after Opt2, we can assume the size of subgraph of the System PDG is $v(i_1+j_1)$, where v is expected to be close to 1. With Opt2, we compare the Bug PDG with w neighbor graphs in the System PDG in Step 3 of CBCD. The time complexity of each comparison will be between $O(w(i_1+j_1+v(i_1+j_1))^2)$ and $O(w(i_1+j_1+v(i_1+j_1))!\ (i_1+j_1+v(i_1+j_1)))$.

Let us compare the time complexity of isomorphism testing without Opt2 with Opt2:

- The best case:
  $O(w(i_1+j_1+v(i_1+j_1))^2)$ vs. $O((i_1+ j_1+ i_2+ j_2)^2)$
- The worst case:
  $O(w(i_1+j_1+v(i_1+j_1))!\ (i_1+j_1+v(i_1+j_1)))$ vs.
  $O((i_1+ j_1+ i_2+ j_2)!\ (i_1+ j_1+ i_2+ j_2))$

Opt2-step2 chooses the vertex kind with the fewest occurrences. So, it reasonable to assume that w is small, namely much less than i_2. In addition, we have observed that the buggy code often includes only a few lines, so we can assume i_1+j_1 is much smaller than i_2+j_2. If the two assumptions stand, the time complexity of comparing the Bug PDG and System PDG with Opt2 will be at least as good as the time complexity of this step without Opt2 in the best case. Even in the worst case, the time complexity with Opt2 will still be better than the one without it, because i_1+j_1 is related to the size of the buggy code, which is often small, while i_2+j_2 is related to the size of the procedure to be compared, which can have hundreds of lines of code.

3) Optimization 3 (Opt3): Exclude Irrelevant PDGs

This optimization discards some parts of the System PDG. The Bug PDG must match within one of the (relatively small) components of the System PDG. More specifically, each node of the Bug PDG must correspond to some node of a System PDG component, so each System PDG component must have as many, or more, nodes of each vertex kind than the Bug PDG does. CBCD discards any System PDG component that does not satisfy this criterion.

For example, suppose the Bug PDG has four nodes of the "expression" vertex kind, two nodes of the "control-point" vertex kind, and two nodes of the "actual-in" vertex kind. If a System PDG component includes four nodes of the "expression" vertex kind, one node of the "control-point" vertex kind, and three nodes of the "actual-in" vertex kind, this System PDG component will be excluded from isomorphism matching, because it has too few nodes of vertex kind "control-point". It therefore cannot be a supergraph of the Bug PDG.

4) Optimization 4 (Opt4): Break Up Large Bug Code Segments

Although most bug segments cover 4 or fewer lines of contiguous code, as shown in Fig. 2, some bug segments are larger. When the buggy code segment is large, Opt1, Opt2, and Opt3 may not be able to improve the performance of the system enough, because:

- When the buggy code segment is large, the Bug PDG will include many vertex kinds. Thus, Opt1 may not be able to prune many edges of the System PDG.
- When the buggy code segment is large, the radius of the Bug PDG will be large. Thus, the sub-graphs of the System PDG after Opt2 will still be large and isomorphism matching will be slow.
- Even if few large Bug PDGs and large System PDGs need to be compared for isomorphism matching, the system will perform very slowly. Thus, Opt3, which reduces the number of comparisons, does not help enough.

To deal with large contiguous buggy code, we implemented a fourth optimization. It is only triggered when the bug has more than 8 lines of contiguous code. The optimization is performed in Step 2 of CBCD and breaks up bug code segments into sub-segments with fewer lines of code. We set two thresholds, which are configurable and default to 4 and 6. The purpose of setting these two thresholds is to split large buggy code segment into smaller sub-segments, and at the same time avoid having too small sub-segments. For a buggy code segment having more than 8 lines of code, CBCD puts the first 4 lines of code in a sub-segment first. If the remaining lines have 6 or fewer lines of code, CBCD does not split it further. Otherwise, CBCD again puts the first 4 lines of the remaining lines in the second sub-segment and reconsiders the remaining lines. CBCD searches for clones of each sub-segment independently, and then merges their corresponding matched clones together. Merging can increase the false positive rate of CBCD, if CBCD merges two unrelated partial matches into a "complete" match that it would never have discovered if using the larger bug PDG. To deal with this issue, CBCD checks the last line of one suspected buggy sub-segment with the first line of another suspected buggy sub-segment to be merged. If the difference is more than 8 lines of code or the two sub-segments are in different files, CBCD assumes that these two code lines are too far apart to be part of clone of a single bug and does not merge them.

IV. Evaluation And Discussion

We wished to answer the following research questions:

- How well can CBCD find cloned buggy code?
- How well does CBCD scale?

A. The Subject Programs

We evaluated CBCD on Git, the Linux kernel, and PostgreSQL. We chose those three systems because:

- They are programmed mainly using C/C++, which means that they can be compiled by CodeSurfer.
- Their revision histories enable us to find buggy code and cloned buggy code for our evaluation.
- Git has more than 100K lines of code, PostgreSQL has more than 300K lines of code, and the Linux kernel has

millions of lines of code, making them a good test of the scalability of CBCD.

B. Evaluation Procedure

1) Oracles for the Evaluation

As discussed in Section III.B, determining true clones of buggy code is undecidable. Our experiments use as an oracle the clones of buggy code that developers identified. It is possible that the developers found only some clones of a given bug, in which case any tool that reported the others would be (incorrectly) considered to suffer false positives.

As described in Section II, we identified buggy code and its clones by searching commit logs and reading code. From these bugs, we chose only those related to C/C++ code, because that is the only type of code that CodeSurfer can compile. We examined all 12 Git bugs and all 22 PostgreSQL bugs from Table III, and we arbitrarily chose 52 (one third of 157) Linux bugs from Table I. We were not able to use all of these bugs: our technique is not applicable when the bug fix adds new code; CBCD only handles C and C++; our processor is 32-bit x86; and in two cases the developers were mistaken in calling two bugs clones, because they refer to completely different functions or data structures (see Table V). After excluding such cases, the evaluation used 5 Git bugs, 14 PostgreSQL bugs, and 34 Linux bugs. A complete list of the bug clones examined in the evaluation is in Appendix A. Appendix D shows the commitment information of the bugs in SCM.

TABLE V. BUGGY CODE THAT PROGRAMMERS CALLED "CLONES" BUT ARE NOT TRUE CLONES

Buggy lines of code	Not identical code under CBCD definition
struct **lock_file** packlock;	struct **cache_file** cache_file;
if (**ahd_match_scb**(ahd, pending_scb, scmd_id(cmd))	if (**ahc_match_scb**(ahc, pending_scb, scmd_id(cmd))

2) Other Code Clone Detectors for Comparison

To compare CBCD with other types of code clone detectors, we also ran Simian v2.3.32 [25] (text-based), CCFinder v10.2.7.3 [1] (token-based), Deckard v1.2.1 [6] (AST-based), and CloneDR v2.2.5 [26] (AST-based) on these 53 bugs.

These code clone detectors favor large cloned code segments rather than small ones. As shown in Fig. 2, cloned bugs are mostly less than 4 lines of code, so we adjusted some parameters to make the code clone detectors work better. For Simian, we set the number of lines of code to be compared for clones to its minimum value, i.e. 2, and used default values for the other parameters. For CCFinder, we set the minimum clone length to be 10 and the minimum TKS to be 1. For Deckard, we set min_tokens to 3, stride to 2, and similiartiy threshold to 0.95. For CloneDR, we set the minimum clone mass to 1, the number of characters per node to 10, number of clone parameters to 5, and similarity threshold to 0.9.

For Simian, CCFinder, and Deckard, the system to be checked for buggy clones is the same file set as CBCD. However, CloneDR failed with parse errors when we input the same file set as for CBCD. To enable a comparison with CBCD, we used a "*slim evaluation*": the "system" input to CloneDR is *only* the files that include the bug and the buggy clones found by CBCD. We additionally commented out lines that CloneDR could not parse. The slim evaluation determines whether CloneDR can find the clones that are identified by CBCD. However, the slim version includes only 2% of the input files and 1% of the lines of code. If CloneDR could run on all files, its false positive rate would be much higher than reported in the slim evaluation.

3) Executing the Tools

The input to each tool is: the file that contains the buggy code (along with the starting and ending lines of the buggy code segment, if the tool accepts it; only CBCD did), plus the system to be checked for buggy clones.

We recorded the execution time of CBCD using the Linux command "time". The evaluation was run on a PC with 4G memory, 3Ghz CPU, and running Ubuntu 10.04.

4) Metrics

A false negative is a clone identified by the developer but not identified by the tool. A false positive is a clone reported by a tool that the developers did not report as buggy.

We count a clone as found if a tool reports a clone pair whose parts are as large as, or larger than, the original buggy code and the developer-identified buggy clone. This metric is very generous to the other code clone tools. CBCD reports clones that have similar size to the buggy code. The other code clone tools report much larger clones, because they are designed for a different purpose: to find large cloned code segments. Often a single result subsumed several of CBCD's results. Such large results would be less useful to a programmer. These issues make a direct comparison of precision and recall, or of the exact number of true and false positives and negatives, misleading. Instead, for each tool, we categorized each of the 53 bugs as follows.

- **N1**: *no false positives, no false negatives.*
- **N2**: *no false positives, some false negatives*.
- **N3**: *some false positives, no false negatives.*
- **N4**: *some false positives, some false negatives.*

C. How Well Can CBCD Find Cloned Buggy Code?

Table VI counts the bugs in each category. Detailed data are shown in Appendix B. CBCD outperforms the other tools in finding buggy clones correctly, i.e., CBCD has the highest number in N1. Deckard performs the worst, partially because it failed with parse errors in 15 out of the 29 N2 cases. Unlike CloneDR, Deckard does not report precisely the location of the parse error. Thus, we could not perform a slim evaluation as with CloneDR.

TABLE VI. COMPARISON WITH OTHER CODE CLONE DETECTORS

	CBCD	Simian	CCFinder	Deckard	CloneDR-slim
N1	36 (68%)	16 (30%)	24 (45%)	14 (26%)	31 (58%)
N2	6 (11%)	36 (68%)	11 (21%)	29 (55%)	14 (26%)
N3	11 (21%)	1 (2%)	12 (23%)	6 (11%)	7 (13%)
N4	0 (0%)	0 (0%)	6 (11%)	4 (8%)	1 (2%)

Researchers categorize code clones into four main types, and so-called "scenarios" subcategorize each type [27]. The distributions of our examined bugs are shown in details in Appendix A and are summarized as follows:

- 51% of duplicated bugs are Type-1: identical code fragments except for variations in whitespace, layout, and comments.
- 24% are in scenarios *a*, *b*, and *c* of Type-2: renaming identifiers or renaming data types and literal values. Most of the variable renaming is renaming of function actual arguments.
- 23% are in scenarios *a* and *b* of Type-3: small deletions or insertions.
- 2% are in scenario *a* of Type-4: reordering of statements.

The 5 tools perform about equally well on Type-1 and Type-2 clones. In theory, AST-based tools could be best on Type-2 clones, but CBCD's text comparisons reduce its false positive rate in practice. CBCD outperforms all the other tools on Type-3 clones; for example, CBCD identifies the code segments shown in Table VII as clones while Simian, CCFinder, Deckard, and CloneDR suffer false negatives.

Unlike text-based, token-based, and AST-based clone detectors, a semantics-based clone detector like CBCD tolerates control-statement replacement. Our 53 examples did not include control-statement replacement (programmers might be less likely to call such code snippets "clones" in the bug tracking system), so we evaluated this claim by artificially modifying the code of a Git clone from a "for" statement to a "while" statement. The modified code is shown in Table VIII. CBCD identified the clone, but Simian, CCFinder, Deckard, and CloneDR did not.

TABLE VII. EXAMPLES OF BUGGY CLONES IDENTIFIED CORRECTLY BY CBCD BUT NOT BY OTHER CODE CLONE DETECTORS

Buggy lines of code	Bug clones
doorbell[0] = cpu_to_be32**((qp ->rq.next_ind << qp- >rq.wqe_shift) | size0)**;	doorbell[0] = cpu_to_be32**(first_i nd << srq->wqe_shift)**;
ret = btrfs_drop_extents(trans, root, inode, start, **aligned_end**, start, &hint_byte);	ret = btrfs_drop_extents(trans, ro ot, inode, file_pos, **file_pos + nu m_bytes,** file_pos, &hint);

TABLE VIII. ORIGINAL CODE VS. CODE AFTER CONTROL REPLACEMENT

Original code	Code after control replacement

for (j = first; j <= last; j++){
 	
struct object_entry *child = objects + deltas[j].obj_no;
if (child->real_type == OBJ_REF_DELTA)
resolve_delta(child, &base_obj, obj->type);
}
	j = first;
	while (j <= last){
	struct object_entry *child = objects + deltas[j].obj_no
	if (child->real_type == OBJ_REF_DELTA)
	resolve_delta(child, &base_obj, obj->type);
	j++; }

The 6 clones out of 53 that are not identified by CBCD, i.e. the false negative cases, are in Table IX. CBCD misses the first three clones because CodeSurfer's PDG does not represent data structures and macros; this is not a reflection on our technique, but on our toolset. CBCD misses the last three clones because they include variable renaming in an expression. When a vertex in the PDG is recognized as "expression", as explained in Section III.C.1, CBCD compares the characters of the expression to avoid false positives.

All 11 bugs for which CBCD reports a false positive are similar: the buggy code is one line of code calling a function, or a few one-line function calls without data/control dependencies among them. For all 11 bugs, Simian, CCFinder, or Deckard either also report a false positive, or else suffer a false negative due to a built-in threshold that prevents them from ever finding any small clone. CloneDR-slim does slightly better, with 2 false negative and 7 false positives. Recall that we used a slim evaluation for CloneDR; if it ran on all files, its false positive rate would be higher.

One example of CBCD's 11 false positives is shown in Table X. Other calls of the same function, such as memset(ib_ah_attr, 0, sizeof param), are returned by CBCD, because it tolerates renaming of actual input and output parameters. However, as mentioned in Section IV.C.3, we count as a false positive any CBCD output that is not yet reported by the developers as buggy. Some of the CBCD-identified clones of the bug code segments might be bugs that have been overlooked by developers. Thus, CBCD's real false positive rate may be lower than Table VI reports.

TABLE IX. FALSE NEGATIVES: BUGGY CODE CLONES THAT ARE NOT IDENTIFIED BY CBCD

The bug fix shown by "diff"
static const struct amd_flash_info jedec_table[] = { - .devtypes = CFI_DEVICETYPE_X16|CFI_DEVICETYPE_X8, - .uaddr = MTD_UADDR_0x0555_0x02AA,
static struct ethtool_ops bnx2x_ethtool_ops = { - .get_link = ethtool_op_get_link,
#define desc_empty(desc) \ - (!((desc)->a + (desc)->b))
- obj = ((struct tag *)obj)->tagged; VS. - object = tag->tagged;
- blue_gain = core->global_gain + core->global_gain * core->blue_bal / (1 << 9); VS. - red_gain = core->global_gain + core->global_gain * core->blue_bal / (1 << 9);
- if (!hpet && !ref1 && !ref2) VS. - if (!hpet && !ref_start && !ref_stop)

TABLE X. EXAMPLES OF FALSE POSITIVES

Buggy code	All identified clones

memset(ib_ah _attr, 0, sizeof *path);	**True positive:**
	memset(ib_ah_attr, 0, sizeof *path);
	False positive:
	memset(best_table, 0, sizeof(best_table));
	memset(best_table_len, 0, sizeof(best_table_len));
	memset(p, 0, padding);
	etc.

Table XI shows another kind of code that might lead to potential false positive reports from CBCD. Fig. 4 shows the PDGs. The two vertexes representing "close()" in Bug PDG

and the four vertexes representing "close()" in System PDG lead to several sub-graph isomorphism relationships between these two PDGs. Thus, CBCD returned several semantically identical correspondences between the buggy code and suspected code. However, all CBCD results point to the same suspected code. CBCD coalesces duplicate results that point to the same code location.

D. How Well Does CBCD Scale to Larger Bugs?

In our experiments, CBCD finished in seconds after CodeSurfer completed. However, this is not a good test of scalability, because the cloned bugs are often platform- or architecture-dependent, in which case the command line (in the developer-supplied Makefile) that compiles them does not compile the whole system.

TABLE XI. BUGGY CODE AND SUSPECTED CODE OF A POTENTIAL FALSE POSITIVE IN GIT

Buggy code	System code

if(pid! = 0){
close(fd[1]);
dup2(fd[0], 0);
close(fd[0]);}
	if(pid! = 0){
	close(fd[1]);
	dup2(fd[0], 0);
	close(fd[0]); }
	close(fd[0]);
	close(fd[1]);

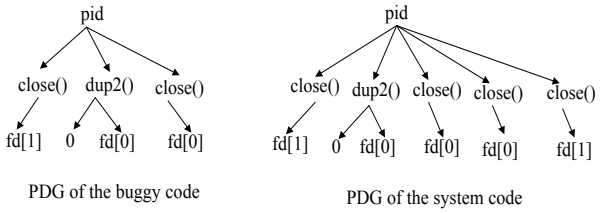

Figure 4. Snippet of the PDG of the buggy and system code in Table XI

To determine how well CBCD works with larger bug segments, we searched the Linux and Git SCM using the key word "duplicate". We chose four of these (non-buggy) code segments from Git and four from Linux. The four Linux code segments are located in subcomponents "net", "fs", "drivers", and "drivers" of Linux of different versions respectively, and we compiled the relevant subcomponent. For Git, we compiled the whole relevant version (Git changed size over time). Table XII gives the results.

Step 1 of CBCD (performed by CodeSurfer, version 2.1) takes a long time if the system is big, but this is done only once and can be reused. We expect CodeSurfer's performance to improve in later versions. Checking for clones of new bugs requires only running Step 2 and 3, which takes only seconds.

The running time of Simian, CCFinder, and Deckard using the same parameter setting as explained in Section IV.B are shown in Table XIII. We could not run CloneDR because of its parse errors.

CBCD is slower than Simian and Deckard if CBCD's preprocessing (Step 1) is included. Considering only the incremental cost of Steps 2 and 3, CBCD is competitive. Setting parameters to let CCFinder detect small clones makes it slower than CBCD, because generating all small clone pairs first, and then searching for clones of a certain code segment, is inherently inefficient. This could be changed, but CBCD is more accurate than the other approaches, regardless of their settings. We believe the cost of undetected bugs makes CBCD worth running even if all steps are required.

E. Performance Improvement Due to the Four Optimizations

We used four optimizations to speed up CBCD. We have examined the unique benefits of a given optimization that are not obtained by other optimizations. For example, to evaluate Opt2, we compared CBCD with Opts 1+3+4 against CBCD with Opts 1+2+3+4.

TABLE XII. RUNNING TIME OF EACH STEP OF CBCD

Id	NLOC / Number of PDG edge		CBCD steps		
			1	*2*	*3*
	Sys.	*Bug*			
Git-1	67K/358K	10/38	6m	**13s**	**5s**
Git-2	75K/441K	4/4	15m	**4s**	**2s**
Git-3	81K/414K	9/39	18m	**9s**	**3s**
Git-4	81K/414K	16/33	18m	**6s**	**2s**
Linux1	170K/1022K	6/70	32m	**15s**	**6s**
Linux2	140K/830K	3/3	25m	**16s**	**4s**
Linux3	363K/1970K	4/4	159m	**39s**	**8s**
Linux4	313K/1645K	3/13	95m	**17s**	**7s**

TABLE XIII. RUNNING TIME OF OTHER CLONE DETECTORS

Id	Simian	CCFinder	Deckard
Git-1	2s	5m	4m
Git-2	2s	6m	5m
Git-3	2s	8m	6m
Git-4	2s	8m	6m
Linux1	6s	63m	8m
Linux2	5s	34m	7m
Linux3	16s	899m	32m
Linux4	13s	623m	24m

The results show that our optimizations can greatly improve the performance of the isomorphism matching by reducing the complexity and number of graphs to be compared. Detailed data are shown in Appendix C .

Opt1, i.e. filtering out the irrelevant edges and vertexes in the System PDG, contributes most to the CBCD performance improvement. Opt1 pruned on average 90% of the edges before the subgraph isomorphism comparison. For the 53 bugs, Opt1 on average improved performance 622 times. However, the variation is high. One case achieved 20237 times performance improvement and another achieved 11890 times performance gain. In one of the four "duplicate code" Linux cases, without Opt1, the execution of the Step 3 of CBCD was aborted (igraph's [16] subgraph isomorphism function reported an out-of-memory error, because the System PDG is too big and too many isomorphic subgraphs are returned).

Opt2, i.e. breaking the System PDG into smaller graphs, improves Step 3 of CBCD by 2 to 3 times. In one case, Opt2 improved performance by 72 times. The performance gain of Opt2 is not significant in other cases, because Opt1 prunes

out most edges of the System PDG. In 90% of our examined cases, the average ratio of size (number of edges and vertexes) of subgraph of the System PDG to size of the Bug PDG, i.e. the "*v*" in the formulas of Section III.C.2, is less than 1.

Opt3, i.e. excluding irrelevant System PDGs, also improves Step 3 of CBCD by 2 to 3 times. As with Opt2, after Opt1 filters out most of the edges of the System PDG, few subgraphs of the System PDGs are left for comparison.

Opt4, i.e. breaking the large bug code segment, is applicable only to three clones that have more than 8 lines of code. In one case, Step 3 of CBCD sped up by 120 times, but the other two showed no significant performance improvement. Examination of these code segments shows that Opt4 can bring significant performance gains when the bug code segment has many vertex kinds, especially vertex kinds such as "actual_in", "actual_out", or "declaration", that are related to procedure parameters or arguments. In such cases, Opt1 cannot filter out many vertexes and edges of the System PDG. On the contrary, if the number of different vertex kinds of the Bug PDG is small, many vertexes and edges of other vertex kinds in System PDGs will be pruned out using Opt1, and Opt2 and Opt3 are also more effective, subsuming the benefits of Opt4.

F. Threats to Validity

1) Threats to Internal Validity

The buggy code used for evaluation consists of real cloned bugs in Git, the Linux kernel, and PostgreSQL, but were not chosen to be representative or comprehensive. We do not know how many cloned bugs these projects really have, but we do know that around 4% of the bugs in a commercial product were duplicates.

2) Threats to External Validity

We tested CBCD only on Git, the Linux kernel, and PostgreSQL. It is possible that other subject programs would have different characteristics. Furthermore, the evaluation considers only 53 cloned bugs in detail, and these were not chosen to be representative.

3) Threats to Construct Validity

To measure the false positive rate of CBCD, we used the clones identified by the developers as an oracle. As mentioned in Section IV.C, the developers might have overlooked some clones, so CBCD's real false positive rate may be lower than reported in this paper.

G. Application Constraints

Although bugs consisting of a one-line function cause false positives in our experiment, and Fig. 2 shows that most code fixes are on one line, this does not limit the applicability of CBCD. In real life, developers can often merge the buggy code line with few lines before or after it, which can be regarded as the context of the buggy code, to make a bigger code segment as the input for CBCD. This may help avoid false positives. We did not perform this in our experiments to avoid evaluation bias.

V. RELATED WORK

Previous code detection methods can be classified into:

- Token-based code clone detecting methods [1, 2] examine token sequence similarities.
- Text [3] or string-based [4] code clone detection methods compare the text or strings in the code.
- Abstract syntax tree (AST) based code clone detection methods [5, 6] match two ASTs to find code clones.
- PDG-based code clone detection tools [7, 8, 9, 10] try to overcome the limitations of the above code clone detectors by comparing the data and control dependence graphs of the code segments.
- Behavior-based code clone detection [32] tries to find code clone based on the execution results of test cases.
- Memory-state-based code clone detection [33] compares the abstract memory states of code.

Most previous code clone detection tools search for large clones for code refactoring or to find plagiarism. Thus, most such tools do not compare small code segments that span only a few lines. For example, PDGs smaller than a certain size are excluded from comparison in [9]. In general, such tools have no knowledge of which segment of code should be the input for clone searching. Thus, some of these tools start with the first line of the system, and extract 10 or 20 lines as input for searching for code clones.

We have identified a new, important use case. CBCD solves a different problem than scanning an entire codebase for plagiarism detection or identifying refactoring opportunities. CBCD is more like an advanced "find" command. The input is a small code segment that includes a few contiguous lines of code (most buggy segments cover only a few contiguous lines of code, unless the bug is caused by missing functionality or a design change). The outputs are all locations of the clones of such a code segment. A user might assume that general code clone detectors would also perform well at detecting clones of buggy code. However, as our evaluation showed, this assumption would be wrong. CBCD outperforms text-based, token-based, and AST-based clone detectors to find cloned buggy code, especially Type-3 and Type-4 clones. We did not compare CBCD with behavior-based clone detectors, because we lack detailed knowledge of the expected dynamic behavior of the buggy code. Memory-state-based clone detectors do not fit the purpose of detecting cloned buggy code.

Unlike generic code clone detectors; CBCD does not generate all code clone pairs in advance. It only searches for clones of a small code segment on demand. The rationale is that people are usually not interested in finding code clones of small code segments to refactor them. However, when they find that a code segment is buggy, they need to find all its clones and fix all of them. As mentioned in Section IV.B and IV.E, searching for clones on demand rather than generating all clone pairs at once makes CBCD more scalable than general clone detectors. But, even if other clone detectors adopted CBCD's incremental approach, CBCD is still more accurate.

CBCD uses PDG-based code clone detection principles to detect clones. PDG-based methods usually face scalability problems in sub-graph isomorphism checking. One proposed solution to improve the performance of PDG-based code

clone detection is to match the PDG back to the AST [10], so that the graph isomorphism problem is simplified into a tree similarity problem. However, such a simplification excludes information for some edges in the PDG and makes the PDG comparison incomplete. Another proposed solution to the scalability problem is to compare the vertex histogram of PDGs first to exclude highly dissimilar PDGs and stop the sub-graph isomorphism matching after the first isomorphism is found [9]. Such a solution is lossy, because a dissimilar vertex histogram between a small PDG and a big PDG does not guarantee that the small PDG will not have a subgraph isomorphism relationship with the large PDG. A PDG-based code clone detector [7] based only on graph isomorphism performed poorly compared to other code clone detectors [30]. CBCD improves the accuracy of PDG-based code clone detection by utilizing the syntax and text information of the buggy code to prune and break the PDG to be compared. Compared to the system in [9], CBCD is less lossy and is more scalable to large PDGs. Yet another proposed solution to the scalability problem is to compare the PDG only within radius 5 of a vertex of "control-point" kind [19]. This is lossy and depends on hard-coded choices of radius and vertex kind; by contrast, our Opt2 is not lossy and is general.

The studies [28, 29] transform the code query into graph reachability patterns and match the patterns in the SDG of the source code. Such a method can potentially be used to detect clones of buggy code. However, developers must manually describe the buggy code using code query language. Compared to these methods, CBCD is easier to use, because it automatically transforms the buggy code into PDG graphs and then matches the buggy PDG with the PDG of the suspected code. Similarity, graph-matching algorithm has been used to match design patterns [34]. However, the algorithm in [34] is not directly applicable since it finds a hard-coded set of design patterns rather than clones of arbitrary bugs. CP-Miner [2] is a code clone detection tool that searches for bugs caused by code copy-paste. CP-Miner can only find "bugs caused when programmers forget to modify identifiers consistently after copy-pasting". The study [31] also compares tokens to search defect clones.

The SecureSync tool [18] is similar to CBCD, i.e. a tool to find duplications of a software vulnerability/bug. To use SecureSync, the clones must be classified into categories I, II, and III first. A category I code clone is due to code copy/paste. For such a code clone, an AST-based method is proposed. A category II code clone is due to function reuse. To detect such a clone, the local PDG around a function call is built and compared. All other code clones are categorized into III without any methods proposed to detect them. Compared to SecureSync, CBCD is easier to use. People do not need to categorize code clone into different categories and treat them differently. For category I code clones, CBCD better tolerates code insertion, deletion, and re-ordering. CBCD can potentially support more kinds of code clone, for example, those in category III of SecureSync. We would like to compare CBCD with SecureSync [18], but according to its authors, SecureSync is not available for public distribution yet. Jiang et al. [20] investigated how to discover clone-related bugs through comparing the nodes in parse trees. In [21], the attributes of edges and nodes of two graphs are extracted to optimize the performance of graph isomorphism comparison for detecting clones of MATLAB/Simulink models. In [22], 17-45% of bug-fixing changes were found to be recurring, and most of them occurred in multiple files at the same revision (i.e. in space). However, this study targets identifying bug clones in object-oriented systems. In [23], a few clone detection algorithms are combined with parallel algorithm to detect buggy inconsistency in a very large system.

VI. Conclusions and Future Work

We have identified a new, important use case for code clone detection (finding buggy clones), motivated its importance in real-world systems, given an algorithm for finding buggy clones, and evaluated its accuracy and performance. Whereas previous work was motivated by code refactoring or plagiarism detection, we focus on detecting cloned buggy code.

The contributions of our work include:

1. We examined real-world bug reports and SCM data, and established that identical (cloned) bugs are a serious problem. In a commercial product line, cloned bugs were common and important, comprising 4% of all bugs.

2. We proposed a methodology for improving system reliability: After a bug is fixed, the programmer should search for other code that behaves similarly to the detected buggy lines. Even if a system has relatively few cloned bugs, finding these bugs is valuable for programmers and can be done relatively accurately and inexpensively.

3. We extended previous PDG-based clone detection algorithms to make them more scalable, by pruning the search space of sub-graph isomorphism matching. Detecting small clones required different algorithms and implementations than previous code detectors, which are less effective in finding bug clones.

4. We implemented our algorithms in a tool, CBCD, that detects possible clones of buggy code by comparing the Bug PDG and the System PDG. The CBCD tool is available on request for research purposes.

5. We evaluated CBCD with known cloned bugs and known cloned lines of code, showing that CBCD is scalable and effective in searching for possible clones of buggy code. Other clone detection tools are less effective for this purpose.

The performance bottleneck of CBCD is CodeSurfer's PDG generation. Future work is to improve performance of this step to make CBCD even more scalable.

Acknowledgments

This work was supported in part by grant #183235/S10 from the Norwegian Research Council, by the JIP partners, and by US NSF grant CCF-1016701.

References

[1] T. Kamiya, S. Kusumoto, and K. Inoue, "CCFinder: a Multilinguistic Token-based Code Clone Detection System for Large Scale Source

Code," IEEE Trans on Software Engineering, vol. 28, no. 7, pp. 654-670, July 2002.

[2] Z. Li, S. Lu, S. Myagmar, and Y. Zhou, "CP-Miner: Finding Copy-Paste and Related Bugs in Large-Scale Software Code," IEEE Trans on Software Engineering, vol. 32, no. 3, pp. 176-192, March 2006.

[3] S. Ducasse, M. Rieger, and S. Demeyer, "A Language Independent Approach for Detecting Duplicated Code," Proc. IEEE intl. conf. on Software Maintenance (ICSM'99), IEEE Press, Sept. 1999, pp. 109-118.

[4] B. S. Baker, "On Finding Duplication and Near-duplication in Large Software Systems," Proc. the Second Working Conference on Reverse Engineering, IEEE Press, July 1995, pp. 86-95.

[5] R. Koschke, R. Falke, and P. Frenzel, "Clone Detection Using Abstract Syntax Suffix Trees," Proc. the 13th Working Conference on Reverse Engineering, IEEE Press, Oct. 2006, pp. 253-262.

[6] L. Jiang, G. Misherghi, Z. Su, and S. Glondu, "DECKARD: Scalable and Accurate Tree-Based Detection of Code Clones," Proc. Intl. conf. on Software Engineering (ICSE'07), IEEE Press, May 2007, pp. 96-105.

[7] J. Krinke, "Identifying Similar Code with Program Dependence Graphs," Proc. the 8th Working Conference on Reverse Engineering (WCRE'01), IEEE Press, Oct. 2001, pp. 301-309.

[8] R. Komondoor and S. Horwitz, "Using Slicing to Identify Duplication in Source Code," Proc. the 8th International Symposium on Static Analysis (SAS' 01), Spring-Verlag Press, July 2001, pp. 40-56.

[9] C. Liu, C. Chen, J. Han, and P. S. Yu, "GPLAG: Detection of Software Plagiarism by Program Dependence Graph Analysis," Proc. 12th ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining, ACM Press, Aug. 2006, pp. 872-881.

[10] M. Gabel, L. Jiang, and Z. Su, "Scalable Detection of Semantic Clones," Proc. Int. Conf. on Software Engineering (ICSE'08), ACM Press, May 2008, pp. 321-330.

[11] A. Chou, J. Yang, B. Chelf, S. Hallem, and D. Engler, "An Empirical Study of Operating Systems Errors," Proc. the 8th ACM Symp. on Operating Systems Principles, ACM Press, Oct. 2001, pp. 73-88.

[12] L. P. Cordella, P. Foggia, C. Sansone, and M. A. Vento, "(Sub)Graph Isomorphism Algorithm for Matching Large Graphs," IEEE Trans on Pattern Analysis and Machine Intelligence, vol. 26, no. 10, pp. 1367-1372, Oct. 2004.

[13] R. C. Read, and D. G. Corneil, "The Graph Isomorphism Disease," Journal of Graph Theory, vol. 1, no. 4, pp. 339–363, Winter 1977.

[14] CodeSurfer: http://www.grammatech.com/products/codesurfer/overview.html

[15] J. Ferrante, K. J. Ottenstein, and J. D. Warren, "The Program Dependence Graph and its Use in Optimization," ACM Trans on Programming Languages and Systems, vol. 9, no. 3, pp. 319-349, July, 1987.

[16] G. Csárdi and T. Nepusz, "The Igraph Software Package for Complex Network Research," InterJournal Complex Systems, 2006, pp. 1695.

[17] B. D. McKay, "Practical Graph Isomorphism," Congressus Numerantium, 30 (1981), pp. 45-87.

[18] N. H. Pham, T. T. Nguyen, H. A. Nguyen, and T. N. Nguyen, "Detection of Recurring Software Vulnerabilities," Proc. Intl. Conf. on Automated Software Engineering (ASE'10), ACM Press, Sept. 2010, pp. 447-456.

[19] R.-Y. Chang, A. Podgurski and J. Yang, "Discovering Neglected Conditions in Software by Mining Dependence Graphs," IEEE Trans on Software Engineering, vol. 34, no. 5, pp. 579-596, Sept. 2008.

[20] L. Jiang, Z. Su, and E. Chiu, "Context-based Detection of Clone-related Bugs," Proc. 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symp. on The foundations of software engineering (ESCE/FSE'07), ACM Press, Sept. 2007, pp. 55-64.

[21] N. H. Pham, H. A. Nguyen, T. T. Nguyen, J. M. Al-Kofahi, and T. N. Nguyen, "Complete and Accurate Clone Detection in Graph-based Models," Proc. Intl. Conf. on Software Engineering (ICSE'09), IEEE Press, May 2009, pp.276-286.

[22] T. T. Nguyen, H. A. Nguyen, N. H. Pham, J. M. Al-Kofahi, and T. N. Nguyen, "Recurring Bug Fixes in Object Oriented Programs," Proc. Intl. Conf. on Software Engineering (ICSE'10), ACM Press, May 2010, pp. 315-324.

[23] M. Gabel, J. Yang, Y. Yu, M. Goldszmidt, and Z. Su, "Scalable and Systematic Detection of Buggy Inconsistencies in Source Code," Proc. ACM intl. conf. on Object Oriented Programming Systems Languages and Applications (OOPSLA'10), ACM Press, Oct. 2010, pp. 175-190.

[24] J. Li, and M. D. Ernst, "CBCD: Cloned Buggy Code Detector," Technical Report UW-CSE-12-03-20, 2012.

[25] Simian- Similarity Analyser: http://www.harukizaemon.com/simian/

[26] CloneDR: http://www.semdesigns.com/Products/Clone/

[27] C. K. Roy, J. R. Cordy, and R. Koschke, "Comparison and Evaluation of Code Clone Detection Techniques and Tools: A Qualitative Approach," Sci. Comput. Program, vol. 74, no. 7, pp. 470-495, May 2009.

[28] X. Wang, D. Lo, J. Cheng, L. Zhang, H. Mei, and J. X. Yu, "Matching Dependence-related Queries in the System Dependence Graph," Proc. Intl. Conf. on Automated Software Engineering (ASE'10), ACM Press, Sept. 2010, pp. 457-466.

[29] M. Martin, B. Livshits, and M. S. Lam, "Finding Application Errors and Security Flaws using PQL: a Program Query Language," Proc. ACM intl. conf. on Object Oriented Programming Systems Languages and Applications (OOPSLA'05), ACM Press, Oct, 2005, pp. 365-383.

[30] S. Bellon, R. Koschke, G. Antoniol, J. Krinke, E. Merlo, "Comparison and Evaluation of Clone Detection Tools," IEEE Trans on Software Engineering, vol. 33, no. 9, pp. 577-591, Sept. 2007.

[31] S. Bazrafshan, R. Koschke, and N. Gode, "Approximate Code Search in Program Histories," Proc. 18th Working Conference on Reverse Engineering, in in press, 2011.

[32] L. Jiang and Z. Su., "Automatic Mining of Functionally Equivalent Code Fragments via Random Testing," Proc. 8th Intl. Symp. on Software Testing and Analysis (ISSTA '09), ACM Press, July 2009, pp. 81-92.

[33] H. Kim, Y. Jung, S. Kim, and K. Yi, "MeCC: Memory Comparison-Based Clone Detector," Proc. 33rd Intl. Conf. on Software engineering (ICSE '11), ACM press, May 2011, pp. 301-310.

[34] N. Tsantalis, A. Chatzigeorgiou, G. Stephanides, and S. T. Halkidis, "Design Pattern Detection Using Similarity Scoring," IEEE Trans. On Software Engineering, vol. 32, no. 11, pp. 896-909, Nov. 2006.

Appendix A: The experimented code by CBCD and evaluation results

Id	Commit id	Buggy code	Clones	Type	Tools results[a]				
					CBCD	Simian[b]	CCFind[c]	Decard[d]	CloneDr[e]
1	postgreSQL-2618fcd	**2618fcd - pg_dump.c: 2672-2675** sprintf(q, "CREATE %s INDEX %s on %s using %s (", (strcmp(indinfo[i].indisunique, "t") == 0) ? "UNIQUE" : "", fmtId(indinfo[i].indexrelname), fmtId(indinfo[i].indrelname), indinfo[i].indamname);	**87d96ed - Pg_dump.c 2673-2676** sprintf(q, "CREATE %s INDEX %s on %s using %s (", (strcmp(indinfo[i].indisunique, "t") == 0) ? "UNIQUE" : "", fmtId(indinfo[i].indexrelname), fmtId(indinfo[i].indrelname), indinfo[i].indamname);	1	N1	N1	N1	N1	N1
2	postgreSQL-161be69	**161be69- Pathnode.c: 336** pathnode->indexqual = NIL;	**1b93294-Pathnode.c: 344** pathnode->indexqual = NIL;	1	N1	N1	N1	N4	N1

3	postgreSQL-dcb09b5	**dcb09b5 - Plperl.c: 2132-2133**
		perm_fmgr_info(typeStruct->typoutput, &(prodesc->arg_out_func[i]));
			dcb09b5 - Plperl.c: 2088-2088
			perm_fmgr_info(typeStruct->typinput, &(prodesc->result_in_func));
			dcb09b5 - Plperl.c: 2720-2720
			perm_fmgr_info(typInput, &(qdesc->arginfuncs[i]));
				3ab	N1	N2	N2	N2	N2

4	postgreSQL-04d976f	**04d975f-date.c: 505** TimeScale = pow(10, typmod);	**C456693- date.c: 505** TimeScale = pow(10, typmod);	1	N1	N1	N3	N2 (parse error)	N1

5	postgreSQL-9dbfcc2	**9dbfcc2- Plperl.c: 758-763**
		for (i = 0; i < tupdesc->natts; i++){/********** Get the attribute name**********/
		attname = tupdesc->attrs[i]->attname.data;
			6d239ee – Plperl.c:758-763
			for (i = 0; i < tupdesc->natts; i++){/********** Get the attribute name**********/
			attname = tupdesc->attrs[i]->attname.data;
				1	N1	N1	N1	N1	N1

6	postgreSQL-d9dddd1	**d9dddd1 -Describe.c: 69-71**
		processNamePattern(&buf, pattern, true, false, "n.nspname", "p.proname", NULL, "pg_catalog.pg_function_is_visible(p.oid)");
			d9dddd1 -Describe.c: 123-125
			processNamePattern(&buf, pattern, false, false,NULL, "spcname", NULL, NULL);
			d9dddd1 -Describe.c: :181-182
			processNamePattern(&buf, pattern, true, false,"n.nspname", "p.proname", NULL, "pg_catalog.pg_function_is_visible(p.oid)");
			d9dddd1 -Describe.c: 435-438
			processNamePattern(&buf, pattern, true, false, "n.nspname", "p.proname", NULL, "pg_catalog.pg_function_is_visible(p.oid)");
			d9dddd1 -Describe.c: 441-443
			processNamePattern(&buf, pattern, true, fals "n.nspname",
				3ab	N1	N2	N2	N2	N2

			"p.proname", NULL,"pg_catalog.pg_function_is_visible(p.oid)");
			d9dddd1 -Describe.c: 447-449
			processNamePattern(&buf, pattern, false, false, "n.nspname", "o.oprname", NULL, "pg_catalog.pg_operator_is_visible(o.oid)");
			d9dddd1 -Describe.c: 478-481
			processNamePattern(&buf, pattern, true,false,"n.nspname", "r.rulename", NULL "pg_catalog.pg_table_is_visible(c.oid)");
			d9dddd1 -Describe.c: 485-487
			processNamePattern(&buf, pattern, false, false, "n.nspname", "t.tgname", NULL, "pg_catalog.pg_table_is_visible(c.oid)");
			d9dddd1 -Describe.c: 535-538
			processNamePattern(&buf, pattern, false, false, "n.nspname", "c.relname", NULL, "pg_catalog.pg_table_is_visible(c.oid)");
			d9dddd1 -Describe.c: 1306-1308
			processNamePattern(&buf, pattern, false, false, NULL, "r.rolname", NULL, NULL);
			d9dddd1 -Describe.c: 1406-1408
			processNamePattern(&buf, pattern, true, false, "n.nspname", "c.relname", NULL, "pg_catalog.pg_table_is_visible(c.oid)");
			d9dddd1 -Describe.c: 1453-1457
			processNamePattern(&buf, pattern, true, false, "n.nspname", "t.typname", NULL, "pg_catalog.pg_type_is_visible(t.oid)");
			d9dddd1 -Describe.c: 1489-1490
			processNamePattern(&buf, pattern, true, false, "n.nspname", "c.conname", NULL "pg_catalog.pg_conversion_is_visible(c.oid)");
			d9dddd1 -Describe.c: 1569-1572
			processNamePattern(&buf, pattern, true, false,NULL, "n.nspname", NULL, NULL);						

7	postgreSQL-0d8e7f6	**0d8e7f6 - Pg_dump.c: 485** fgets(username, 9, stdin);	**087eb4c-Pg_dump.c: 506** fgets(password, 9, stdin);	2ab	N3	N2	N3	N3	N3
8	postgreSQL-8474600	**8474600 - Int8.c:309** return (*val1 > *val2) ? val1 : val2;	**8474600 - Int8.c: 328** return (*val1 < *val2) ? val1 : val2;	3ab	N1	N2	N2	N1	N2
9	postgreSQL-19dacd4	**19dacd4-Timestamp.c: 3536-3537** case DTK_YEAR: result = tm->tm_year;	**f2c064a –Timestamp.c:3263-3264** case DTK_YEAR: result = tm->tm_year;	1	N1	N2	N3	N2 (parse error)	N2

10	postgreSQL-db6df0c	**3b6bf0c- Postmaster.c : 1741**
		if (BgWriterPID != 0)
		kill(BgWriterPID, SIGTERM);
			3b6bf0c- Postmaster.c: 1775
			if (BgWriterPID != 0)
			kill(BgWriterPID, SIGTERM);
			3b6bf0c- Postmaster.c : 1809
			if (BgWriterPID != 0)
			kill(BgWriterPID, SIGTERM);
			3b6bf0c- Postmaster.c : 1857
			if (BgWriterPID != 0)
			kill(BgWriterPID, SIGQUIT);
				2ab	N1	N2	N1	N2 (parse error)	N1

11	postgreSQL-dcb09b5	**dcb09b5 - Int_bool.c: 199** if (lenstack && (stack[lenstack - 1] == (int4) '&' || stack[lenstack - 1] == (int4) '!')){	**dcb09b5 - ltxtquery_io.c: 244** if (lenstack && (stack[lenstack - 1] == (int4) '&' || stack[lenstack - 1] == (int4) '!'))	1	N1	N1	N3	N2	N3
12	postgreSQL-6666185	**6666185 - Pgstattuple.c:258** scan = heap_beginscan(rel, SnapshotAny, 0, NULL);	**689d02a- index.c: 2009** scan = heap_beginscan(heapRelation, /* relation */ snapshot,/* seeself */ 0, /* number of keys */ NULL); /* scan key */	3ab	N3	N2	N4	N4 (parse error)	N2

13	postgreSQL-54bce38	**54bce38- Setrefs.c: 93**
		fix_opids((Node *) ((IndexScan *) plan)->indxqualorig);
		plan->subPlan = nconc(plan->subPlan,
		pull_subplans((Node *) ((IndexScan *) plan)->indxqual));
			54bce38- Setrefs.c: 106
			fix_opids((Node *) ((MergeJoin *) plan)->mergeclauses);
			plan->subPlan = nconc(plan->subPlan, pull_subplans((Node *) ((MergeJoin *) plan)->mergeclauses));
			54bce38- Setrefs.c: 145
			fix_opids(((Result *) plan)->resconstantqual);
			plan->subPlan = nconc(plan->subPlan, pull_subplans(((Result *) plan)->resconstantqual));
			54bce38- Setrefs.c: 113-115
			fix_opids((Node *) ((HashJoin *) plan)->hashclauses);
			plan->subPlan =nconc(plan->subPlan, pull_subplans((Node *) ((HashJoin *) plan)->hashclauses));
			54bce38- Setrefs.c: 145-147
			fix_opids(((Result *) plan)->resconstantqual);
			plan->subPlan =nconc(plan->subPlan, pull_subplans(((Result *) plan)->resconstantqual));
			54bce38- Setrefs.c: 168
			fix_opids((Node *) plan->qual);
			plan->subPlan =nconc(plan->subPlan, pull_subplans((Node *) plan->targetlist));
				3ab	N1	N2	N2	N2	N2

14	postgreSQL-f4d108a	**f4d108a - Parse_func.c: 902-909**
		else if (nmatch == nbestMatch){
		last_candidate->next = current_candidate;
		last_candidate = current_candidate;
		ncandidates++;}
			42af563 - parse_oper.c:220-229
			else if (nmatch == nbestMatch) {last_candidate->next = current_candidate;
			last_candidate = current_candidate;
			ncandidates++; }
			/* otherwise, don't bother keeping this one... */
			else
				1	N1	N2	N1	N2	N1

		else last_candidate->next = NULL;	last_candidate->next = NULL; **42af563 - parse_oper.c 273-280** else if (nmatch == nbestMatch) { last_candidate->next = current_candidate; last_candidate = current_candidate; ncandidates++;} else **42af563 - parse_func.c 802-805** else if (nmatch == nbestMatch) { last_candidate->next = current_candidate; last_candidate = current_candidate; ncandidates++;} else						
15	git-a3eb250	**a3eb250-clone-pack.c: 154-157** If(!pid){ close(fd[1]); dup2(fd[0], 0); close(fd[0]);	**a3eb250 – fetch-pack.c: 97-105** If(!pid){ close(fd[1]); dup2(fd[0], 0); close(fd[0]);	1	N1	N2	N1	N2	N1
16	git-b3118bd	**b3318bd-sha1_file.c:1360 - 361** if (st == Z_BUF_ERROR && (stream.avail_in || !stream.avail_out)) break;	**b3318bd-sha1_file.c: 1599-1600** if (st == Z_BUF_ERROR && (stream.avail_in || !stream.avail_out)) break;	1	N1	N1	N1	N1	N1
17	git-da0204d	**da0204d - builtin-fetch.c: 265** commit = lookup_commit_reference(rm->old_sha1);	**42a3217: builtin-fetch--tool.c: 148** commit = lookup_commit_reference(sha1);	3ab	N3	N2	N3	N4	N2

18	git-cd03eeb	**cd03eeb-transport-helper.c:41**
		write_in_full(helper->in, buf.buf, buf.len);
			cd03eeb-transport-helper.c: 61
			write_in_full(data->helper->in, "\n", 1);
			cd03eeb-transport-helper.c:87
			write_in_full(helper->in, buf.buf, buf.len);
				3ab	N3	N2	N4	N4	N4

19	git-013aab	**013aab-a3eb250-commit.c:55** if (obj->type == tag_type) obj = ((struct tag *)obj)->tagged;	**013aab-a3eb250 – rev-list.c: 370** if (object->type == tag_type) { object = tag->tagged;	3ab	N2	N2	N2	N2	N2
20	linux-5bb1ab	**5bb1ab-exthdrs.c:691** IP6_INC_STATS_BH(ipv6_skb_idev(skb), IPSTATS_MIB_INHDRERRORS);	**5bb1ab-exthdrs.c: 698** IP6_INC_STATS_BH(ipv6_skb_idev(skb), IPSTATS_MIB_INHDRERRORS); **5bb1ab-exthdrs.c: 703** IP6_INC_STATS_BH(ipv6_skb_idev(skb), IPSTATS_MIB_INHDRERRORS); **5bb1ab-exthdrs.c: 709** IP6_INC_STATS_BH(ipv6_skb_idev(skb), IPSTATS_MIB_INTRUNCATEDPKTS);	2ab	N3	N2	N3	N3	N1
21	linux-590929f	**590929f-mt9v001.c: 203-205** blue_gain = core->global_gain + core->global_gain * core->blue_bal / (1 << 9);	**590929f-mt9v001.c: 205-206** red_gain = core->global_gain + core->global_gain * core->blue_bal / (1 << 9);	2ab	N2	N2	N1	N1	N1
22	linux-9378b	**9278b63-Tsc.c :467**	**9278b63-Tsc.c :938**	2ab	N2	N2	N3	N2 (parse	N2

	63	if (!hpet && !ref1 && !ref2)	if (!hpet && !ref_start && !ref_stop)					error)	
23	linux-fe1cbab	**fe1cbab-transfd.c:919** err = sock_create_kern(PF_UNIX, SOCK_STREAM, 0, &csocket);	**fe1cbab-transfd.c:957** err = sock_create_kern(PF_UNIX, SOCK_STREAM, 0, &csocket);	2ab	N1	N2	N4	N1	N1
24	linux-d89197c	**d8917c-Eeprom_def.c: 1065** case 2: scaledPower -= REDUCE_SCALED_POWER_BY_TWO_CHAIN;	**333ba73-ar9003_eeprom.c: 4647** case 2: scaledPower -= REDUCE_SCALED_POWER_BY_TWO_CHAIN;	1	N1	N1	N1	N2 (parse error)	N1

25	linux7-cab758e	**Cab758e: tcp_ipv4.c: 1591**
		if (nsk != sk) {
		if (tcp_child_process(sk, nsk, skb)) {
			Cab758e: tcp_ipv6.c:1646
			if(nsk != sk) {
			if (tcp_child_process(sk, nsk, skb))
				1	N1	N1	N1	N1	N2

26	linux-0029227	**0029227 - xhci.c: 515** xhci_cleanup_msix(xhci);	**0029227 - xhci.c: 551** xhci_cleanup_msix(xhci);	1	N3	N3	N3	N3	N3
27	linux-713b3c9	**713b3c9: Ixgbe_main.c: 3731** hw->mac.ops.setup_sfp(hw);	**713b3c9: Ixgbe_main.c: 5971** hw->mac.ops.setup_sfp(hw);	1	N1	N2	N3	N2 (parse error)	N1

28	linux-52534f2	**cfi_cmdset_0002.c: 714**
		map_write(map, cfi->sector_erase_cmd, chip->in_progress_block_addr);
			cfi_cmdset_0001.c: 818
			map_write(map, CMD(0x70), adr);
			cfi_cmdset_0001.c:816
			map_write(map, CMD(0xd0), adr);
				3ab	N3	N2	N4	N2	N2

29	linux-dcace06	**dcaece6 - dw_mmc:1205** tasklet_schedule(&host->tasklet);	**dcaece6 - dw_mmc:1214** tasklet_schedule(&host->tasklet);	1	N3	N2	N3	N3	N3
30	linux-a57ca04	**a57ca04 - jedec_probe.c: 1159-1160** .devtypes = CFI_DEVICETYPE_X16|CFI_DEVICETYPE_X8, .uaddr = MTD_UADDR_0x0555_0x02AA, /* ?? ?? */	**f636ffb - jedec_probe.c 1464-1465** .devtypes = CFI_DEVICETYPE_X16 |CFI_DEVICETYPE_X8, .uaddr = MTD_UADDR_0x0AAA_0x0555, }	2ab	N2	N2	N2	N2	N3
31	linux-ff0ac74	**ff0ac74-bnx2x_main.c:10037** .get_link = ethtool_op_get_link,	**0f77ca9 – bnx2.c:7395** .get_link = ethtool_op_get_link,	1	N2	N2	N2	N2 (parse error)	N1
32	linux-5153f7	**5153f7 - asm-i386/processor.h: 32** (!((desc)-	**5153f7 - asm-x86_64/processor.h :35** (!((desc)->a + (desc)->b))	1	N2	N1	N2	N2	N1

		>a + (desc)->b))							

33	linux-8bea867	**8bea867 - drm_fb_helper.c: 57-63**
		static int my_atoi(const char *name)
		{ int val = 0;
		for (;; name++) {
		switch (*name) {
		case '0' ... '9':
		val = 10*val+(*name-'0');
		break;
		default:
		return val; }
		} }
			8bea867 - modedb.c: 409-414
			static int my_atoi(const char *name)
			{ int val = 0;
			for (;; name++) {
			switch (*name) {
			case '0' ... '9':
			val = 10*val+(*name-'0');
			break;
			default:
			return val; } } }
				1	N1	N1	N1	N1	N1

34	linux-ea2d8b5	**ea2d8b5- iwl3945-base.c: 5771** ieee80211_notify_mac(priv->hw, IEEE80211_NOTIFY_RE_ASSOC);	**ea2d8b5 – iwl-agn.c: 2093** ieee80211_notify_mac(priv->hw, IEEE80211_NOTIFY_RE_ASSOC);	1	N1	N2	N3	N2 (parse error)	N1
35	linux-c9a2c46	**c9a2c46 - w83781d.c: 1369-1372** - if (!request_region(**res->start, W83781D_EXTENT, "w83781d"**)) {	**c9a2c46 - lm78.c: 657-660** if (!request_region(**res->start, LM78_EXTENT, "lm78"**)) {	2ab	N1	N2	N1	N1	N1

36	linux-d555009	**d555009-visor.c: 609-611**
		result = usb_submit_urb(**priv->bulk_read_urb, GFP_ATOMIC**);
		if (result) dev_err(&port->dev, "%s failed submitting read urb, error %d\n",
			d555009- opticon.c: 167-168
			result = usb_submit_urb(priv->bulk_read_urb, GFP_KERNEL);
			if (result) dev_err(&port->dev, "%s failed resubmitting read urb, error %d\n",__func__, result);
			d555009- opticon.c: 327-329
			result = usb_submit_urb(**priv->bulk_read_urb, GFP_ATOMIC**);
			if (result) dev_err(&port->dev, "%s failed submitting read urb, error %d\n",
				2ab	N1	N2	N3	N2	N1

37	linux-9601e3f	**9601e3f - inode.c 236-237** ret = btrfs_drop_extents(**trans, root, inode, start, aligned_end, start, &hint_byte**);	**9601e3f - inode.c : 1457-1458** ret = btrfs_drop_extents(**trans, root, inode, file_pos, file_pos + num_bytes, file_pos, &hint**);	3ab	N1	N2	N4	N2	N2

38	linux-2567d71	**2567d71 - rcuclassic.c 141-146**
		rdp = &__get_cpu_var(**rcu_data**);
		*rdp->nxttail = head;
		rdp->nxttail = &head->next;
		if (unlikely(++rdp->qlen > qhimark)) {
		rdp->blimit = INT_MAX;
		force_quiescent_state(rdp, &rcu_ctrlblk); }}
			2567d71 - rcuclassic.c 177-183
			rdp = &__get_cpu_var(**rcu__bg_data**);
			*rdp->nxttail = head;
			rdp->nxttail = &head->next;
			if (unlikely(++rdp->qlen > qhimark)) {
			rdp->blimit = INT_MAX;
			force_quiescent_state(rdp, &rcu_ctrlblk); }}
				2ab	N1	N2	N1	N1	N1

39	linux-3976ae6	**3976ae6 - rt2400pci.c :296-298**
		
		rt2x00_set_field32(&reg, CSR14_TSF_COUNT, 1);
		rt2x00_set_field32(&reg, CSR14_TBCN,
		(conf->sync == TSF_SYNC_BEACON));
		rt2x00_set_field32(&reg, CSR14_BEACON_GEN, 0);
			3976ae6 - rt2500pci.c: 299-302
			
			rt2x00_set_field32(&reg, CSR14_TSF_COUNT, 1);
			rt2x00_set_field32(&reg, CSR14_TBCN,
			(conf->sync == TSF_SYNC_BEACON));
			rt2x00_set_field32(&reg, CSR14_BEACON_GEN, 0);
				1	N1	N1	N1	N2 (parse error)	N1

40	linux-c09c518	**c09c518 - w83627hf.c 1332-1335** if (reg & 0xff00) { outb_p(W83781D_REG_BANK, data->addr + W83781D_ADDR_REG_OFFSET); outb_p(reg >> 8, data->addr + W83781D_DATA_REG_OFFSET) }	**c09c518 - w8362hf.c: 1347** if (reg & 0xff00) { outb_p(W83781D_REG_BANK, data->addr + W83781D_ADDR_REG_OFFSET); outb_p(**0**, data->addr + W83781D_DATA_REG_OFFSET); } **393cdad - w8362hf.c: 1422** if (reg & 0xff00) { outb_p(W83781D_REG_BANK, data->addr + W83781D_ADDR_REG_OFFSET); outb_p(reg>>8, data->addr + W83781D_DATA_REG_OFFSET); } **393cdad - w8362hf.c: 1437** if (reg & 0xff00) { outb_p(W83781D_REG_BANK, data->addr + W83781D_ADDR_REG_OFFSET); outb_p(**0**, data->addr + W83781D_DATA_REG_OFFSET); }	2ab	N1	N2	N2	N2 (parse error)	N1
41	linux-b45bfcc	**1c27327 - qp.c :1503:** memset(ib_ah_attr, 0, sizeof *path);	**b5bfcc - mthca_qp.c: 402** memset(ib_ah_attr, 0, sizeof *path);	1	N3	N2	N1	N3	N3
42	linux-34cc560	**34cc560 - tcp_output.c: 481** th->window = htons(tp->rcv_wnd);	**34cc560 - tcp_output.c: 2160** th->window = htons(req->rcv_wnd);	1	N3	N2	N4	N2 (parse error)	N1

43	linux-efbfe96c	**efbfe96c - vmscan.c: 976-977**
		if (zone->prev_priority > priority)
		zone->prev_priority = priority;
			efbfe96c - vmscan.c: 1187-1188
			if (zone->prev_priority > priority)
			zone->prev_priority = priority;
				1	N1	N2	N1	N3	N1

44	linux-093beac	**093beac - mthca_qp.c 1730-1738:**
		for (nreq = 0; wr; ++nreq, wr = wr->next) {
		if (unlikely(nreq == MTHCA_TAVOR_MAX_WQES_PER_RECV_DB)) {
		nreq = 0;
		doorbell[0] = cpu_to_be
			093beac - mthca_srq.c: 493-500
			for (nreq = 0; wr; ++nreq, wr = wr->next) {
			if (unlikely(nreq == MTHCA_TAVOR_MAX_WQES_PER_RECV_DB)) {
			nreq = 0;
			doorbell[0] = cpu_to_be32(**first_ind << srq->wqe_shift**);
				3ab	N1	N2	N2	N2 (parse error)	N2

		32((**qp->rq.next_ind << qp->rq.wqe_shift**) | size0);
		doorbell[1] = cpu_to_be32(qp->qpn << 8);
		wmb();
		mthca_write64(doorbell,
		dev->kar + MTHCA_RECEIVE_DOORBELL,
		MTHCA_GET_DOORBELL_LOCK(&dev->doorbell_lock));
			doorbell[1] = cpu_to_be32(srq->srqn << 8);
			wmb();
			mthca_write64(doorbell,
			dev->kar + MTHCA_RECEIVE_DOORBELL,
			MTHCA_GET_DOORBELL_LOCK(&dev->doorbell_lock));						

45	linux-a6230af	**a6230af - readdir.c: 217-218**
		if(cifs_sb->mnt_cifs_flags & CIFS_MOUNT_NO_BRL)
		tmp_inode->i_fop->lock = NULL;
			a6230af - readdir.c: 334-335
			if(cifs_sb->mnt_cifs_flags & CIFS_MOUNT_NO_BRL)
			tmp_inode->i_fop->lock = NULL;
				1	N1	N1	N1	N1	N1

46	linux-c87e34e	**c87e34e - sg.c 1863-1865**
		if (res > 0)
		for (j=0; j < res; j++)
		page_cache_release(pages[j]);
			c87e34e - st.c: 4509-4511
			if (res > 0) {
			for (j=0; j < res; j++)
			page_cache_release(pages[j]);
			}
				3ab	N1	N1	N1	N2 (parse error)	N2

47	linux-5917583	**5917583-mremap.c 145-147**
		if (pfn_valid(pte_pfn(pte))
		&&
		pte_page(pte) == ZERO_PAGE(old_addr))
		pte = pte_wrprotect(mk_pte(ZERO_PAGE(new_addr), new_vma->vm_page_prot));
			676d55a-mremap.c 145-147
			if (pfn_valid(pte_pfn(pte)) &&
			pte_page(pte) == ZERO_PAGE(old_addr))
			pte = pte_wrprotect(mk_pte(ZERO_PAGE(new_addr), new_vma->vm_page_prot));
				1	N1	N1	N1	N1	N1

48	linux-19147bb	**19147bb - e1000/e1000_main.c: 2052-2057**
		if (buffer_info->dma) {
		pci_unmap_page(adapter->pdev,
		buffer_info->dma,
		buffer_info->length,
		PCI_DMA_TODEVICE);
		buffer_info->dma = 0;
		}
			19147bb - e1000e/net_dev.c: 569-571
			if (buffer_info->dma) {
			pci_unmap_page(adapter->pdev, buffer_info->dma,
			buffer_info->length, PCI_DMA_TODEVICE);
			buffer_info->dma = 0;
			}
				1	N1	N2	N1	N1	N1

49	linux-4c25a2c	**4c25a2c - dmar.c: 755-759**
		if (non_present_entry_flush) {
		if (!cap_caching_mode(iommu->cap))
		return 1;
		else
		did = 0;
		}
			4c25a2c - intel-iommu.c: 916-920
			if (non_present_entry_flush) {
			if (!cap_caching_mode(iommu->cap))
			return 1;
			else
			did = 0;
			}
				1	N1	N1	N1	N1	N1

50	linux-529ed80	**529ed80 - i810-i2c.c: 48-50** i810_writel(mmio, chan->ddc_base, (state ? SCL_VAL_OUT : 0) | **SCL_DIR** | SCL_DIR_MASK | SCL_VAL_MASK); i810_readl(mmio, chan-	**529ed80 - i810-i2c.c: 59-60** i810_writel(mmio, chan->ddc_base, (state ? SDA_VAL_OUT : 0) | **SDA_DIR** | SDA_DIR_MASK | SDA_VAL_MASK); i810_readl(mmio, chan->ddc_base); /* flush posted	2ab	N1	N2	N1	N2 (parse error)	N1

		>ddc_base); /* flush posted write */	write */						

51	linux-3083e83	**3083e83- iwl-core.c 1145-1148**
		priv->tx_power_next = tx_power;
		if (test_bit(STATUS_SCANNING, &priv->status) && !force) {
		IWL_DEBUG_INFO(priv, "Deferring tx power set while scanning\n");
		return 0;
			efe1cf0- iwl-core.c 1193-1196
			priv->tx_power_next = tx_power;
			if (test_bit(STATUS_SCANNING, &priv->status) && !force) {
			IWL_DEBUG_INFO(priv, "Deferring tx power set while scanning\n");
			return 0;
				1	N1	N1	N1	N2	N1

52	linux-78794b2	**78794b2- main.c: 63-69** INIT_RADIX_TREE(&mapping->page_tree, GFP_ATOMIC); spin_lock_init(&mapping->tree_lock); **spin_lock_init(&mapping->i_mmap_lock);** **INIT_LIST_HEAD(&mapping->private_list);** **spin_lock_init(&mapping->private_lock);** INIT_RAW_PRIO_TREE_ROOT(&mapping->i_mmap); INIT_LIST_HEAD(&mapping->i_mmap_nonlinear);	**78794b2- page.c: 498-504** INIT_RADIX_TREE(&mapping->page_tree, GFP_ATOMIC); spin_lock_init(&mapping->tree_lock); **INIT_LIST_HEAD(&mapping->private_list);** **spin_lock_init(&mapping->private_lock);** **spin_lock_init(&mapping->i_mmap_lock);** INIT_RAW_PRIO_TREE_ROOT(&mapping->i_mmap); INIT_LIST_HEAD(&mapping->i_mmap_nonlinear);	4a	N1	N2	N1	N2	N1

53	linux-c594d88	**c594d88 - ops_address.c: 233**
		gfs2_holder_init(ip->i_gl, LM_ST_SHARED, GL_ATIME|GL_AOP, **&gh**);
			c594d88 - ops_address.c: 295
			gfs2_holder_init(ip->i_gl, LM_ST_SHARED, LM_FLAG_TRY_1CB|GL_ATIME|GL_AOP, &gh);
			c594d88 - ops_address.c: 369
			gfs2_holder_init(ip->i_gl, LM_ST_EXCLUSIVE, GL_ATIME|GL_AOP, &ip->i_gh);
				2ab	N3	N2	N2	N2	N3

D1	git-d53fe81	**d53fe81- archive-tar.c: 281-292**
		if (args->baselen > 0 && args->base[args->baselen - 1] == '/') {
		char *base = xstrdup(args->base);
		int baselen = strlen(base);
		while (baselen > 0 && base[baselen - 1] == '/')
		base[--baselen] = '\0';
		write_tar_entry(args->tree->object.sha1, "", 0, base, 040
			d53fe81 – builtin-checkout.c: 327-338
			if (args->baselen > 0 && args->base[args->baselen - 1] == '/') {
			char *base = xstrdup(args->base);
			int baselen = strlen(base);
			while (baselen > 0 && base[baselen - 1] == '/')
			base[--baselen] = '\0';
			write_zip_entry(args->tree->object.sha1, "", 0, base, 040777, 0, NULL);
			free(base);
			}
			read_tree_recursive(args-						

		777,
		0, NULL);
		free(base);
		}
		read_tree_recursive(args->tree, args->base, args->baselen, 0,
		args->pathspec, write_tar_entry, NULL);
			>tree, args->base, args->baselen, 0,
			args->pathspec, write_zip_entry, NULL
			);						

D2	git-3fe2a89	**3fe2a89 – builtin-commit.c: 940-974**
		if (s.relative_paths)
		s.prefix = prefix;
		if (s.use_color == -1)
		s.use_color = git_use_color_default;
		if (diff_use_color_default == -1)
		diff_use_color_default = git_use_color_default;
			3fe2a89 – builtin-commit.c: 982-986
			if (s.relative_paths)
			s.prefix = prefix;
			if (s.use_color == -1)
			s.use_color = git_use_color_default;
			if (diff_use_color_default == -1)
			diff_use_color_default = git_use_color_default;						

D3	git-e923eae	**e923eae-builtin-checkout.c:138-145**
		if (!hashcmp(sha1, null_sha1)) {
		mm->ptr = xstrdup("");
		mm->size = 0;
		return;
		}
		mm->ptr = read_sha1_file(sha1, &type, &size);
		if (!mm->ptr || type != OBJ_BLOB)
		die("unable to read blob object %s", sha1_to_hex(sha1));
		mm->size = size;
			e923eae-merge-recursive.c: 608-615
			if (!hashcmp(sha1, null_sha1)) {
			mm->ptr = xstrdup("");
			mm->size = 0;
			return;
			}
			mm->ptr = read_sha1_file(sha1, &type, &size);
			if (!mm->ptr || type != OBJ_BLOB)
			die("unable to read blob object %s", sha1_to_hex(sha1));
			mm->size = size;						

D4	git-e923eae-2	**e923eae – connect.c: 414-425**
		if (host[0] == '[') {
		end = strchr(host + 1, ']');
		if (end) {
		*end = 0;
		end++;
		host++;
		} else
		end = host;
		} else
		end = host;
		colon = strchr(end, ':');
		if (colon) {
		*colon = 0;
		port = colon + 1;
		}
			e923eae – connect.c: 179-191
			if (host[0] == '[') {
			end = strchr(host + 1, ']');
			if (end) {
			*end = 0;
			end++;
			host++;
			} else
			end = host;
			} else
			end = host;
			colon = strchr(end, ':');
			if (colon) {
			*colon = 0;
			port = colon + 1;
			}						

D5	linux-23edcc4	**23edcc4-ipv4/tcp_input.c: 4904-4909**
		if (tcp_fast_parse_options(skb, th, tp) && tp->rx_opt.saw_tstamp &&
			23edcc4-ipv4/tcp_input.c: 5280-5285
			if (tcp_fast_parse_options(skb, th, tp) && tp->rx_opt.saw_tstamp && tcp_paws_discard(sk, skb)) {
			if (!th->rst) {
			NET_INC_STATS_BH(sock_net(s						

		tcp_paws_discard(sk, skb)) {
		if (!th->rst) {
		NET_INC_STATS_BH(sock_net(sk), LINUX_MIB_PAWSESTABREJECTED);
		tcp_send_dupack(sk, skb);
		goto discard;
		}
			k), LINUX_MIB_PAWSESTABREJECTED);
			tcp_send_dupack(sk, skb);
			goto discard;
			}						

D6	linux-ec33679	**ec33679- dcache.c: 357-360**
		if (IS_ROOT(dentry))
		parent = NULL;
		else
		parent = dentry->d_parent;
			ec33679- dcache.c: 599-602
			if (IS_ROOT(dentry))
			parent = NULL;
			else
			parent = dentry->d_parent;						

D7	linux-2644487	**2644487 – intel_overlay.c: 442-445**
		obj = overlay->vid_bo->obj;
		
		i915_gem_object_unpin(obj);
		
		drm_gem_object_unreference(obj);
			2644487 – intel_overlay.c: 860-862
			obj = overlay->vid_bo->obj;
			i915_gem_object_unpin(obj);
			
			drm_gem_object_unreference(obj);						

D8	linux-a4e77d0	**a4e77d0 – netdev.c: 4672-4674**
		if (le16_to_cpu(buf) & (1 << 0)) {
		e_warn("Warning: detected DSPD enabled in EEPROM\n");
		}
			a4e77d0 – netdev.c: 4678-4680
			if (le16_to_cpu(buf) & (3 << 2)) {
			
			e_warn("Warning: detected ASPM enabled in EEPROM\n");
			}						

a. There are four categories of the results as follows

- **N1**: *no false positives,no false negatives.*
- **N2**: *no false positives, some false negatives*.
- **N3**: *some false positives, no false negatives.*
- **N4**: *some false positives, some false negatives.*

b. Siminan (versn 2.3.32) parameters: threshold = 2. Others are default values
c. CCFinder (version beta 10.2.7.3) Minimum Clone length = 10, Minimum TKS = 1
d. Decard (Version 1.2.1), parameter, min_tokens = 3, stride=2, similarity = 0.95
e. CloneDr (Evaluation version www.semdesigns.com/Products/Clone/) parameters: Similiarity threshold = 0.9; Number of clone parameters = 5; Maximum parameter count=5; Minimum clone mass = 1; Number of characters per node = 10; Starting depth = 1

Appendix B. Running time of CBCD, 3000HZ, 4G, Linux Ubuntu 10.04

Id	Commit id	Compile command[a]	Sys. Size[b]			Running time step 1			Step2[c]	Step3[c]
			NLOC	PDG vertex	PDG edge	Codesurfer compile	Extract PDGs	Check PDG sub-comp		
1	postgreSQL-2618fcd	S: postgreSQL-87d96ed Make all. The "Make" file has been slimed to compile only files in the "bin" component B: postgreSQL-2618fcd Make all. The "Make" file has been slimed to compile only files in the "bin" component	14594	16678	38997	26s	3.9s	0.7s	1s	0.3s
2	postgreSQL-161be69	S: postgreSQL-1b93294 Make all. The "Make" file has been slimed to compile only files in the "lexverify" and "backend" component B: postgreSQL-161be69 Make all. The "Make" file has been slimed to compile only files in the "lexverify" and "backend" component.	134064	197838	463362	13m23s	58s	2m48s	6s	2s
3	postgreSQL-dcb09b5	S&B: postgreSQL-dcb0965 Make all. The "Make" file has been slimed to compile only files in the "plperl" component	13836	30179	66376	52s	45s	3.5s	1s	0.4s
4	postgreSQL-04d976f	S: postgreSQL-c456693 Mak all. The "Make" file has been slimed to compile only files in the "backend" component B: postgreSQL-04d976f Mak all. The "Make" file has been slimed to compile only files in the "backend" component	173070	249251	577127	16m32s	1m26s	3m49s	8s	3s
5	postgreSQL-9dbfcc2	S: postgreSQL-6d239ee Mak all. The "Make" file has been slimed to compile only files in the "pl" component B: postgreSQL-9dbfcc2 Mak all. The "Make" file has been slimed to compile only files in the "pl" component	14259	4308	7945	14s	4s	0.1s	0.2s	0.2s
6	postgreSQL-d9dddd1	S&B: postgreSQL-d9dddd1 Mak all. The "Make" file has been slimed to compile only files in the "bin" component	56263	43701	107890	1m40s	5s	7.9s	2s	44s
7	postgreSQL-0d8e7f6	S: postgreSQL-087eb4c Mak all. The "Make" file has been slimed to compile only files in the "bin" component B: postgreSQL-0d8e7f6 Mak all. The "Make" file has been slimed to compile only files in the "bin" component	19078	18768	43893	29s	6.5s	1s	0.6s	0.4s
8	postgreSQL-8474600	S&B: postgreSQL-8474600 Make all. The "Make" file has been slimed to compile only files in the "backend" component	139795	199560	467561	18m46s	48s	2m50s	4s	2s
9	postgreSQL-19dacd4	S: postgreSQL-f2c064a Make all. The "Make" file has been slimed to compile only files in the "backend" component	227360	375304	812543	72m33s	5m10s	7m40s	7s	4s

		B: postgreSQL-19dacd4 Make all. The “Make” file has been slimed to compile only files in the “backend” component								
10	postgreSQL-db6df0c	S: postgreSQL-3b6bf0c Make all, however. The “Make” file has been slimed to compile only files in the “backend” component B: postgreSQL-db6df0c Make all, however. The “Make” file has been slimed to compile only files in the “backend” component	221783	378741	821737	135m	4m43s	8m26s	12s	4.5s
11	postgreSQL-dcb09b5	S&B: postgreSQL-dcb09b5 Make ./contrib/ltree/ltxtquery_io.o	4478	895	2208	9s	1.9s	0.1s	0.1s	0.1s
12	postgreSQL-6666185	S: postgreSQL-689d02a Make all. The “Make” file has been slimed to compile only files in the “backend” component B: postgreSQL-6666185 Make all. The “Make” file has been slimed to compile only files in the “backend” component	54040	90197	216941	3m42s	21s	9s	2.7s	1s
13	postgreSQL-54bce38	S&B: postgreSQL54bce38 Make ./backend/optimizer/plan/setrefs.o	237	374	938	2s	0.8s	0.1s	0.1s	0.1s
14	postgreSQL-f4d108a	S: postgreSQL-42af56e Make all. The “Make” file has been slimed to compile only files in the “backend” component B: postgreSQL-f4d108a Make all. The “Make” file has been slimed to compile only files in the “backend” component	64040	87433	226345	3m43s	20s	41s	2s	1s
15	git-a3eb250	S: git-a3eb250 make git-fetch-pack B: git-a3eb250 make git-clone-pack	6485	13 664	31 120	31s	3s	0.7s	0.3s	0.2s
16	git-b3118bd	S&B: git-b3118bd make git	37494	166 875	383 615	20m15s	24s	20s	3s	2s
17	git-da0204d	S&B: git-da0204d make git	55286	145 845	333 780	9m33s	19s	52s	4s	2s
18	git-cd03eeb	S&B: git-cd03eeb make git	44090	166 512	382 805	12m23s	18s	22s	6s	4s
19	git-013aab	S&B: git-013aab make git-rev-list	8730	14962	34065	46s	1.6s	0.7s	0.4s	0.2s
20	linux-5bb1ab	S&B: linux-2.6-5bb1ab make ./net/ipv6/exthdrs.o	20040	24330	58752	1m27s	21s	1.9s	0.9s	0.2s
21	linux-590929f	S&B: linux-590929f make ./drivers/media/video/me9v011.o	17093	25397	60139	1m23s	24s	1.9s	0.6s	0.5s
22	linux-9378b63	S&B: linux-9378b63 make ./arch/x86/kernel/tsc.o	19774	25904	61355	1m25s	27s	2s	0.7s	0.4s
23	linux-fe1cbab	S&B: linux-fe1cbab make ./net/9p/trans_fd.o	20758	26649	62947	1m32s	29s	2s	0.9s	0.4s
24	linux-d89197c	S: linux-2.6-333ba73 make ./drivers/net/wireless/ath9k/ar9003_eeprom.o B: linux-2.6-d89197c Make ./drivers/net/wireless/ath9k/eeprom_def.o	22657	26918	64600	1m23s	29s	2s	0.7s	0.5s

25	linux-cab758e	S&B: linux-cab758e make ./net/ipv4/tcp_ipv4.o	34100	36245	86495	1m45s	50s	3s	1.3s	0.6s
26	linux-0029227	S&B: linux-0029227 make ./drivers/usb/host/xhci.o	21307	30695	72520	1m31s	35s	2s	1.1s	0.5s
27	linux-713b3c9	S&B: linux-713b3c9 make ./drivers/net/ixgbe_main.o	34035	36077	86669	1m44s	40s	3s	0.9s	0.5s
28	linux-52534f2	S&B: linux-52534f2 make ./drivers/mtd/chips/cif_cmdset_001.o	21998	29051	70056	1m26s	30s	2s	0.9s	0.5s
29	linux-dcace06	S&B: linux-dcace06 make ./drivers/mmc/host/dw_mmc.o	20565	27690	65281	1m28s	29s	2s	1s	0.5s
30	linux-a57ca04	S: linux- a57ca04 make ./drivers/mtd/chips/jedec_probe.o B: linux-f636ffb make ./drivers/mtd/chips/jedec_probe.o	18864	21636	52292	36s	1.6s	0.6s	0.3s	0.1s
31	linux-ff0ac74	S: linux-ff0ac74 make ./drivers/net/bnx2x_main.o B: linux-0f77ac9 make ./drivers/net/bnx2.o	40078	35044	83241	45s	2s	1.5s	0.7s	0.3s
32	linux- 5153f7	S&B: linux- 5153f7 make ./arch/i386/kernel/process.o	7375	8594	19149	40s	1s	0.6s	0.2s	0.1s
33	linux-8bea867	S: linux-8bea867 make drivers/video/modedb.o B: linux-8bea867 make drivers/gpu/drm/drm_fb_helper.o	17894	24 446	58 836	1m30s	3s	2s	0.06s	0.3s
34	linux-ea2d8b5	S: linux-ea2d8b5 make drivers/net/wireless/iwlwifi/iwl-agn.o B: linux-ea2d8b5 make drivers/net/wireless/iwlwifi/iwl3945-base.o	30407	29 302	69 367	1m19s	3s	2s	0.07s	0.04s
35	linux-c9a2c46	S: linux-c9a2c46 make drivers/hwmon/lm78.o B: linux-c9a2c46 make drivers/hwmon/w83781d.o	16965	22 717	56 345	1m38s	3s	2s	0.08s	0.3s
36	linux-d555009	S: linux-d555009 make drivers/usb/serial/opticon.o B: linux-d555009 make drivers/usb/serial/visor.o	19294	24 902	59 373	1m23s	7s	2s	0.9s	0.3s
37	linux-9601e3f	S&B: linux-9601e3f make fs/btrfs/inode.o	27516	34 074	81 865	1m27s	9s	3s	1s	0.6s
38	linux-2567d71	S&B: linux-2567d71 make kernel/rcuclassic.o	16170	22 857	56 629	1m36s	9s	2s	1s	0.4s
39	linux-3976ae6	S: linux-3976ae6 make drivers/net/wireless/rt2x00/rt2500pci.o B: linux-3976ae6 make drivers/net/wireless/rt2x00/rt2400pci.o	18295	25 879	60 099	1m16s	8s	2s	1.6s	0.6s
40	linux-c09c518	S&B: linux-c09c518 make drivers/hwmon/w83627hf.o	7007	12 452	28 032	49s	4s	2s	0.4s	0.2s
41	linux-b45bfcc	S: linux-b45bfcc linux-1c27cb7/make drivers/infiniband/hw/mlx4/qp.o B: linux-b45bfcc drivers/infiniband/hw/mthca/mthca_qp.o	9870	13 990	33 362	49s	5s	2s	1.5s	0.4s
42	linux-34cc560	S&B: linux-34cc560 make net/ipv4/tcp_output.o	51599	98 426	239 895	49s	18s	10s	2.7s	1.3s

43	linux-efbfe96c	S&B: linux-efbfe96c make mm/vmscan.o	12434	11 039	25 107	41s	4s	2s	0.3s	0.1s
44	linux-093beac	S: linux-093beac make drivers/infiniband/hw/mthca/mthca_qp.o B: linux-093beac make drivers/infiniband/hw/mthca/mthca_srq.o	8695	6 481	17 020	26s	4s	2s	0.8s	0.3s
45	linux-a6230af	S&B: linux-a6230af make fs/cifs/readdir.o	3592	1 745	4 144	25s	1s	1s	0.07s	0.02s
46	linux-c87e34e	S: linux-c87e34e make drivers/scsi/st.o B: linux-c87e34e make drivers/scsi/sg.o	11500	6221	17260	27s	2s	1s	0.3s	0.1s
47	linux-5917583	S: /linux-676d55a/make mm/ B: make mm/	32372	52 384	123 130	1m32s	9s	2s	1.3s	0.7s
48	linux-19147bb	S: linux-19147bb make drivers/net/e1000e/netdev.o B: linux-19147bb make drivers/net/e1000/e1000_main.o	29291	32 388	77 348	1m37s	8s	2s	0.8s	0.04
49	linux-4c25a2c	S: linux-4c25a2c make drivers/pci/intel-iommu.o B: linux-4c25a2c make drivers/pci/dmar.o	20889	26 632	62 840	1m16s	2s	2s	0.6s	0.3s
50	linux-529ed80	S&B: linux-529ed80 make drivers/video/i810/	15774	23 310	55 491	1m12s	2s	2s	0.9s	0.4s
51	linux-3083e83	S: linux-efe1cf0 make drivers/net/wireless/iwlwifi/iwl-core.o B: linux-3083e83 make drivers/net/wireless/iwlegacy/iwl-core.o	11158	26 493	62 373	1m30s	2s	3s	0.8s	0.3s
52	linux-78794b2	S: linux-78794b2 make fs/nilfs2/page.o B: linux-78794b2 make fs/gfs2/main.o	20276	25 328	59 730	1m20s	7s	3s	1.9s	0.6s
53	linux-c594d88	S&B: linux-c594d88 make fs/gfs2/ops_address.o	9024	10 023	22 441	39s	2s	8s	0.2s	0.1s
D1	git-d53fe81	S&B: git-d53fe81 make git	67294	157 328	358 183	5m36s	23s	56s	13s	5s
D2	git-3fe2a89	S&B: git-3fe2a89 make git	75414	196 871	441 496	13m32s	44s	1m2s	4s	2s
D3	git-e923eae	S&B: git-e923eae make git	80944	181 424	414 780	16m18s	40s	1m4s	9s	3s
D4	git-e923eae-2	S&B: git-e923eae-2 make git	80944	181 424	414 780	16m18s	41s	1m4s	6s	2s
D5	linux-23edcc4	S&B: linux-23edcc4 make net/	170021	4 33 970	1 022 407	23m9s	1m25	8m	15s	6s
D6	linux-ec33679	S&B: linux-ec33679 make fs/	140325	367300	830575	18m35s	1m8s	6m36s	16s	3.7s
D7	linux-2644487	S&B: linux-2644487 make drivers/	363440	859 526	1 970 025	126m38s	3m32s	30m9s	39s	8s
D8	linux-a4e77d0	S&B: linux-a4e77d0 make drivers/	313044	705 068	1 645 538	72m29	2m49s	21m17s	17s	7s

a. In some cases, the buggy code and its clones stay in different files or even different versions of the system. Thus, we sometimes have to compile the file including the buggy code and the file including the clones separately to generate Bug PDG and System PDG respectively. The “S” is the compiling command we used to generate the System PDG and the “B” is the command for generating the Bug PDG. To compile Linux to include both the buggy code

and its clones is tricky because:

- We always run the “make defconfig” command first the set the value of the variables in the compiling configuration file
- When we run “make” command afterwards, only the files that are related to our hardware, i.e. the ones identified though “make defconfig” will included for compiling. Due to the hardware setting of our experiment machine, many of the files containing the buggy code or the files containing the system code cannot be included for compiling if we just simply run the command “make” or “make drivers”. For example, when we run the command “make driver” the file “drivers/net/e1000e/netdev.o” will not be included in the compiled result, because we do not have the hardware related to this file. Fortunately, you can always compile only one “.o” file of Linux. Thus, we only compiled the “.o” file of the buggy code and its clones. This works for the code in the *drivers*, *net*, *fs*, and *mm* modules. However, for code in the *arch* or *kernel* modules, specific hardware is needed to compile .c/.cpp to generate even the “.o” code. For example, the ARM processor is needed to compile an “.o” file that is related to the ARM processor. Thus, we have to exclude some cases, which need specific hardware installation, in our experiment.
- For d1 to d8 cases to test the performance of the CBCD, we got many cases return when we searched the Linux and Git SCM using the keyword “duplicate”. Thus, we managed to find files, which include a certain code segment and its duplications, that can be compiled using command like “make drivers”
- In some cases e.g. postgreSQL-2618fcd , the buggy code and its clones are in different commits/versions of the project. That is why another commit IDs are different for the suspected code and for the buggy code.

b. The system size includes the number of different vertexes and the edges in the System PDG.

c. This running time reflect the time that CBCD need to search for the clone of a bug.

Appendix C. Improvement of the optimizations, 3000HZ, 4G, Linux Ubuntu 10.04

Id	Commit Id	Running time with all optimizations[a]		Without Opt1, i.e. prune irrelevant edges[b]				Without Opt2, break system PDG[c]	Without Opt3, exclude irrelevant system PDG[d]	Without Opt4, split Bug PDG[e]
		Step 2	Step 3	Time for step3 (Times step3 time with all opts)	Edge size before prune	Edge size after prune	Edge size reduction rate	Time for step3 (Times step3 time with all opts)	Time for step3 (Times step3 time with all opts)	Time for step3 (Times step3 time with all opts)
1	postgreSQL-2618fcd	1s	0.3s	1.4s (4)	38997	11692	70%	0.3s (1)	0.3s (1)	N/A
2	postgreSQL-161be69	6s	2s	24s (12)	463362	2	99%	2s (1)	2s (1)	N/A
3	postgreSQL-dcb09b5	1s	0.4s	0.8s (2)	66376	8261	88%	0.4s (1)	0.4s (1)	N/A
4	postgreSQL-04d976f	8s	3s	5.6s (2)	577127	92115	84%	3s (1)	3s (1)	N/A
5	postgreSQL-9dbfcc2	0.2s	0.2s	0.1s (0.5)	7945	42	99%	0.1s (0.5)	0.1s (0.5)	N/A
6	postgreSQL-d9dddd1	2s	44s	2m43s (4)	107890	18743	83%	2m26s(4)	1m11s(2)	N/A
7	postgreSQL-0d8e7f6	0.6s	0.4s	0.4s (1)	43893	5603	87%	0.3s (1)	0.2s (0.5)	N/A
8	postgreSQL-8474600	4s	2s	6.3s (3)	467561	4002	99%	2s (1)	4s (2)	N/A
9	postgreSQL-19dacd4	7s	4s	9s (2)	812543	3	99%	4s (1)	4s (1)	N/A
10	postgreSQL-db6df0c	12s	4.5s	35s (8)	821737	95006	88%	4.5s (1)	4.6s (1)	N/A
11	postgreSQL-dcb09b5	0.1s	0.1s	0.1s (1)	2208	41	98%	0.1s (1)	0.1s (1)	N/A
12	postgreSQL-6666185	2.7s	1s	1.5s (2)	216941	42052	81%	1.1s (1)	1.1s (1)	N/A
13	postgreSQL-54bce38	0.1s	0.1s	0.1s (1)	938	287	69%	0.1s (1)	0.1s (1)	N/A
14	postgreSQL-f4d108a	2s	1s	1.9s (2)	226345	2354	99%	1s (1)	1s (1)	N/A
15	git-a3eb250	0.3s	0.2s	8s (40)	31120	324	99 %	0.55s (3)	0.4s (1)	N/A
16	git-b3118bd	3s	2s	3s (1.5)	383615	8	99 %	3.5s (2)	3s (2)	N/A
17	git-da0204d	4s	2s	9s (5)	333780	22	99 %	6.5s (3)	4s (2)	N/A
18	git-cd03eeb	6s	4s	12s (3)	382805	248417	35 %	24s (6)	9.8s (2)	N/A
19	git-013aab	0.4s	0.2s	0.3s(2)	34065	409	99%	0.2 (1)	0.3(2)	N/A
20	linux-5bb1ab	0.9s	0.2s	1s (5)	24579	19634	20%	0.5s (3)	0.3s (1)	N/A
21	linux-590929f	0.6s	0.5s	0.4s (1)	60139	3106	95%	0.3s (1)	0.3s (1)	N/A
22	linux-9378b63	0.7s	0.4s	0.5s (1)	61355	851	99%	0.3s (1)	0.3s (1)	N/A
23	linux-fe1cbab	0.9s	0.4s	0.4s (1)	62947	10685	83%	0.3s (1)	0.3s (1)	N/A
24	linux-d89197c	0.7s	0.5s	0.4s (1)	64600	1	99%	0.3s (1)	0.4s (1)	N/A
25	linux7-cab758e	1.3s	0.6s	0.6s (1)	86495	14209	84%	0.5s (1)	0.4s (1)	N/A
26	linux-0029227	1.1s	0.5s	0.6s (1)	72520	14354	80%	0.5s (1)	0.4s (1)	N/A
27	linux-713b3c9	0.9s	0.5s	0.5s (1)	86669	202	99%	0.4s (1)	0.4s (1)	N/A
28	linux-52534f2	0.9s	0.5s	0.6s (1)	70056	6571	91%	0.4s (1)	0.4s (1)	N/A
29	linux-dcace06	1s	0.5s	0.8s (1)	65281	12656	81%	0.4s (1)	0.5s (1)	N/A
30	linux- a57ca04	0.3s	0.1s	0.1s (1)	52292	0	100%[f]	0.1s (1)	0.1s (1)	N/A
31	linux-ff0ac74	0.7s	0.3s	0.3s(1)	83241	0	100%	0.3s (1)	0.3s (1)	N/A
32	linux- 5153f7	0.2s	0.1s	0.1s (1)	19149	0	100%	0.1s (1)	0.1s (1)	N/A
33	linux-8bea867	0.06s	0.3s	0.7s (2)	58836	79	99 %	0.7s (2)	0.6s (2)	N/A
34	linux-ea2d8b5	0.07s	0.04s	0.9s (23)	69367	2	99 %	**2.9s (72)**	0.06s (2)	N/A

35	linux-c9a2c46	0.08s	0.3s	**26s (130)**	56345	22	99 %	1.5s (5)	1.8s (6)	N/A
36	linux-d555009	0.9s	0.3s	**101m11s (20237)**	59373	430	99 %	1.8s (6)	1.8s (6)	N/A
37	linux-9601e3f	1s	0.6s	49s (82)	81865	214	99 %	4.5s (7)	0.9s (1)	N/A
38	linux-2567d71	1s	0.4s	**79m16s (11890)**	56629	134	99 %	3.5s (8)	1.8s (4)	N/A
39	linux-3976ae6	1.6s	0.6s	17s (28)	60099	13454	77 %	6.9s (4)	6.6s (4)	N/A
40	linux-c09c518	0.4s	0.2s	2.5s (12)	28032	12	99 %	0.8s (4)	0.8s (4)	N/A
41	linux-b45bfcc	1.5s	0.4s	2s (5)	33362	120	99 %	6.2s (15)	1.4 (3)	N/A
42	linux-34cc560	2.7s	1.3s	5s (4)	239895	3623	98 %	4s (3)	3.7s (3)	N/A
43	linux-efbfe96c	0.3s	0.1s	0.3s (3)	25107	2	99 %	0.6s (6)	0.2s (2)	N/A
44	linux-093beac	0.8s	0.3s	5.1s (15)	17020	14276	16 %	5.6s (15)	3s (10)	N/A
45	linux-a6230af	0.07s	0.02s	0.1s (3)	4144	2	99 %	0.24s (12)	0.05s (2)	N/A
46	linux-c87e34e	0.3s	0.1s	0.3s (3)	17260	239	98 %	0.7s (7)	0.2s (2)	N/A
47	linux-5917583	1.3s	0.7s	2.2s (4)	123130	520	99 %	2s (3)	1.4s (2)	N/A
48	linux-19147bb	0.8s	0.04	**4.5s (110)**	77348	5	99 %	2s (50)	0.7s (17)	N/A
49	linux-4c25a2c	0.6s	0.3s	19s (63)	62840	5	99 %	0.6s (2)	0.6s (2)	N/A
50	linux-529ed80	0.9s	0.4s	3s (8)	55491	16	99 %	4s (10)	1.8s (4)	N/A
51	linux-3083e83	0.8s	0.3s	14s (47)	62373	57	99 %	0.9s (3)	0.6s (2)	N/A
52	linux-78794b2	1.9s	0.6s	**1m43s (172)**	59730	131	99 %	7.2s (12)	3s (5)	N/A
53	linux-c594d88	0.2s	0.1s	0.45s (5)	22441	220	99 %	1.1s (10)	0.2s (2)	N/A
D1	git-d53fe81	13s	5s	6m7s (74)	358183	47099	86 %	54s (10)	12s (2)	**10m2s (120)**
D2	git-3fe2a89	4s	2s	14s (7)	441496	21	99 %	11s (5)	6s (3)	N/A
D3	git-e923eae	9s	3s	3m49s (76)	414780	23843	94 %	17s (6)	11s (3)	3s (1)
D4	git-e923eae-2	6s	2s	15s (7)	414780	12212	97 %	8s (4)	7s (4)	2s (1)
D5	linux-23edcc4	15s	6s	**Aborted**	1022407	292	99 %	6s (1)	7.8s (1)	N/A
D6	linux-ec33679	16s	3.7s	24.8s (8)	830575	12	99 %	8.8 (2)	6.8s (2)	N/A
D7	linux-2644487	39s	8s	55s (7)	1970025	94	99 %	39s (5)	31s (4)	N/A
D8	linux-a4e77d0	17s	7s	**15m38s (134)**	1645538	3125	99 %	7.7s (1)	7.7 (1)	N/A

a. These are the running time of the step 2 (prune the system PDG) and step 3 (subgraph testing) of CBCD with all Opts included.

b. Here are the data of CBCD step 3 running time without Opt1, i.e. without pruning the system PDG before subgraph testing. The data in the parentheses are the ratio between the running time here and the running time with all Opts included. The data here also show how much percentages of the edges are pruned out before in step 2 before step 3.

c. The data here show the CBCD step 3 running time without Opt2, i.e. without breaking the system PDG into smaller ones using neighbor graphs. The results show that the running time here is often 2-3 times the CBCD step 3 running times with the Opt2.

d. The data here show the CBCD step 3 running time without Opt3, i.e. excluding the system neighbor subgraphs that are irrelevant. The results show that the running time here is often 2-3 times the CBCD step 3 running times with the Opt3.

e. The data here show the CBCD step 3 time without splitting the bug code segments, when the code segments are big. Here we chose just three cases for experiment, because these the bug code segments of these three cases have more than 8 lines of code.

f. In the linux-a57ca04 case, the edge reduction ratio is 100%, because no Bug PDG was generated, due to Codersurfer cannot catch the buggy code information.

Appendix D. Commit information of the evaluated code

Id	Commit id	Commit info. of the bug	Commit info. of clones
1	postgreSQL-2618fcd	http://git.postgresql.org/gitweb/?p=postgresql.git;a=commit;h=88800aac14c54f595d288be0e1fac8720f5f5b5d Ok. BTW Mr. Kataoka who is maintaing Japanese version of PostgreSQL ODBC driver have found a bug in 6.3.2 pg_dump and have made patches. I confirmed that the same bug still exists in the current source tree. So I made up patches based on Kataoka's. Here are some explanations. o fmtId() returns pointer to a static memory in it. In the meantime there is a line where is fmtId() called twice without saving the first value returned by fmtId(). So second call to fmtId() will break the first one. o findTableByName() looks up a table by its name. if a table name containins upper letters or non ascii chars, fmtId() will returns a name quoted in double quotes, which will not what findTableByName() wants. The result is SEG fault. -- Tatsuo Ishii t-ishii@sra.co.jp sprintf(q, "CREATE %s INDEX %s on %s using %s (", (strcmp(indinfo[i].indisunique, "t") == 0) ? "UNIQUE" : "", - fmtId(indinfo[i].indexrelname), - fmtId(indinfo[i].indrelname), + id1, + id2, indinfo[i].indamname);	http://git.postgresql.org/gitweb/?p=postgresql.git;a=commit;h=b542fa1a6e838d3e32857cdfbe8aeff940a91c74 The same file, but different version
2	postgreSQL-161be69	http://git.postgresql.org/gitweb/?p=postgresql.git;a=commit;h=7d572886d631011114787caa31b90ecaf52c17db Fix coredump seen when doing mergejoin between indexed tables, for example in the regression test database, try select * from tenk1 t1, tenk1 t2 where t1.unique1 = t2.unique2; 6.5 has this same bug ... pathnode->indexkeys = index->indexkeys; - pathnode->indexqual = NIL;	http://git.postgresql.org/gitweb/?p=postgresql.git;a=commit;h=275a1d054e72b35bfd98c9731e51b2961ab8dbf5 Undo Jan's typo that broke regress.sh's detection of system type name. The same file pathnode->indexkeys = index->indexkeys; 344 pathnode->indexqual = NIL;
3	postgreSQL-dcb09b5	http://git.postgresql.org/gitweb/?p=postgresql.git;a=commitdiff;h=7748e9e7e5aef280bea4e204017e8ac7dca14177;hp=7c0c9b3ccec4718c1c7cef7b5282fd56b727d965 pltcl, plperl, and plpython all suffer the same bug previously fixed in plpgsql: they fail for datatypes that have old-style I/O functions due to caching FmgrInfo structs with wrong fn_mcxt lifetime. Although the plpython fix seems straightforward, I can't check it here since I don't have Python installed --- would someone check it? - fmgr_info(typeStruct->typinput, &(prodesc->result_in_func));	Three different files in the same submission Because there are too many bugs in the plperl.c of the current version, it is impossible to compile it. So, we use the commit in 11 to run the test. There, the buggy function name has been changed to perm_fmgr_in fo(..).

4	postgreSQL-04d976f	http://git.postgresql.org/gitweb/?p=postgresql.git;a=commit;h=1392cbd0ed97f1bf956d4aa2cc4325f9a6418e8b AdjustTimeForTypmod has the same bug ...	http://git.postgresql.org/gitweb/?p=postgresql.git;a=commit;h=64dff0beac3c76dd7035bfaa2e4357aa4798cc96 Fix some problems in new variable-resolution-timestamp code.

5	postgreSQL-9dbfcc2	http://git.postgresql.org/gitweb/?p=postgresql.git;a=commit;h=fe055e928095658eb2a8cd52ff32f090720de3de
		
		Looks like plperl has same bug as pltcl.
		
		for (i = 0; i < tupdesc->natts; i++)
		{
		+ /* ignore dropped attributes */
		+ if (tupdesc->attrs[i]->attisdropped)
		+ continue;
			http://git.postgresql.org/gitweb/?p=postgresql.git;a=commit;h=9dbfcc22613379e89283282db5cd616898bf6e4f
			
			Fix some problems with dropped columns in pltcl functions.

6	postgreSQL-d9dddd1	http://git.postgresql.org/gitweb/?p=postgresql.git;a=commitdiff;h=39ed8c4049c2900af348059efe362becdcaf9eb1;hp=d9dddd11000a1f97ad521af7466cc3fb89666997 pg_dump as well as psql. Since psql already uses dumputils.c, while there's not any code sharing in the other direction, this seems the easiest way. Also, fix misinterpretation of patterns using regex | by adding parentheses (same bug found previously in similar_escape()). This should be backpatched.	Same commitment
7	postgreSQL-0d8e7f6	http://git.postgresql.org/gitweb/?p=postgresql.git;a=commitdiff;h=3ac9688ae80ec6bcbb9bdafa8ef30eadc8c6dd6e;hp=087eb4cd1a1faba95699b642883ba588bf709157 prompt_for_password code that psql does. We fixed psql a month or two back to permit usernames and passwords longer than 8 characters. I propagated the same fix into pg_dump. Tom Lane printf("Username: "); - fgets(username, 9, stdin);	http://git.postgresql.org/gitweb/?p=postgresql.git;a=commit;h=cb7cbc16fa4b5933fb5d63052568e3ed6859857b Hi, here are the patches to enhance existing MB handling. This time I have implemented a framework of encoding translation between the backend and the frontend. Also I have added a new variable setting command: SET CLIENT_ENCODING TO 'encoding'; Other features include: Latin1 support more 8 bit cleaness See doc/README.mb for more details. Note that the pacthes are against May 30 snapshot. Tatsuo Ishii
8	postgreSQL-8474600	http://git.postgresql.org/gitweb/?p=postgresql.git;a=commitdiff;h=1d1cf38c0d02908e3c6520dab94c878947ca8152;hp=84746009c2e5686217679ccaae6ed2a18164d37c rather than reusing the input storage. Also made the same fix to int8smaller(), though there wasn't a symptom, and went through and verified that other pass-by-reference data types do the same thing. Not an issue for the by-value types. return (*val1 > *val2) ? val1 : val2;	The same commitment
9	postgreSQL-19dacd4	http://git.postgresql.org/gitweb/?p=postgresql.git;a=commit;h=c584103f56040f1c3d2d125256b005ff09c4d94e Patch of 2004-03-30 corrected date_part(timestamp) for extracting	http://git.postgresql.org/gitweb/?p=postgresql.git;a=commit;h=fd071bd478f489c81208029265e1fef954a9b5fa

		the year from a BC date, but failed to make the same fix in date_part(timestamptz). case DTK_YEAR: - result = tm->tm_year;	Fix to_char for 1 BC. Previously it returned 1 AD. Fix to_char(year) for BC dates. Previously it returned one less than the current year. Add documentation mentioning that there is no 0 AD.
10	postgreSQL-db6df0c	http://git.postgresql.org/gitweb/?p=postgresql.git;a=commitdiff;h=0cb117eb33558bc779df833480958a97227dcbc2;hp=3b6bf0c07d49b1172ee0326e3e06583068fa305d Repair some problems in bgwriter start/stop logic. In particular, don't allow the bgwriter to start before the startup subprocess has finished ... it tends to crash otherwise. (The same problem may have existed for the checkpointer, I'm not entirely sure.) Remove some code that was redundant because the bgwriter is handled as a member of the backend list.	The same file
11	postgreSQL-dcb09b5	http://git.postgresql.org/gitweb/?p=postgresql.git;a=commit;h=abc10262696e53773c9a8c9f279bbd464b464190 After parsing a parenthesized subexpression, we must pop all pending ANDs and NOTs off the stack, just like the case for a simple operand. Per bug #5793. Also fix clones of this routine in contrib/intarray and contrib/ltree, where input of types query_int and ltxtquery had the same problem. Back-patch to all supported versions.	The same commitment
12	postgreSQL-6666185	http://git.postgresql.org/gitweb/?p=postgresql.git;a=commit;h=b775d93acb961ceea1371d6c724317e1ea6f3242 Fix pgstat_heap() to not be broken by syncscans starting from a block higher than zero. Same problem as just detected in CREATE INDEX CONCURRENTLY. - scan = heap_beginscan(rel, SnapshotAny, 0, NULL);	http://git.postgresql.org/gitweb/?p=postgresql.git;a=commitdiff;h=d3b1b1f9d8d70017bf3e8e4ccf11b183d11389b9;hp=689d02a2e9c56dbad3982a440278e937fd063260 Fix CREATE INDEX CONCURRENTLY so that it won't use synchronized scan for its second pass over the table. It has to start at block zero, else the "merge join" logic for detecting which TIDs are already in the index doesn't work. Hence, extend heapam.c's API so that callers can enable or disable syncscan. (I put in an option to disable buffer access strategy, too, just in case somebody needs it.) Per report from Hannes Dorbath.
13	postgreSQL-54bce38	http://git.postgresql.org/gitweb/?p=postgresql.git;a=commit;h=2190cf2926961b43e7c2d4415db23c1ccf4c026e Repair bug reported by ldm@apartia.com: Append nodes, which don't actually use their targetlist, are given a targetlist that is just a pointer to the first appended plan's targetlist. This is OK, but what is not OK is that any sub-select expressions in said tlist were being entered in the subPlan lists of both the Append and the first appended plan. That led to two startup and two shutdown calls for the same plan node at exec time, which led to crashes. Fix is to not generate a list of subPlans for an Append node. Same problem and fix apply to other node types that don't have a real, functioning targetlist: Material, Sort, Unique, Hash.	The same commitment

14	postgreSQL-f4d108a	http://git.postgresql.org/gitweb/?p=postgresql.git;a=commitdiff;h=5253c518aef4c906dc6c922c51c2d77b0a78bf75;hp=f4d108a25747754b5d265b12ef32c791ab547782 agg_select_candidate, which could cause them to keep more candidates than they should and thus fail to select a single match. I had previously fixed the identical bug in oper_select_candidate, but didn't realize that the same error was repeated over here. Also, repair func_select_candidate's curious notion that it could scribble on the input type-OID vector. That was causing failure to apply necessary type coercion later on, leading to malfunction of examples such as select date('now').	http://git.postgresql.org/gitweb/?p=postgresql.git;a=commitdiff;h=5adebf83b6cffbf4133ff97dbe6d5da0ff59bff1;hp=42af56e1ead3306d2c056ff96ea770e4eee68e9d Clean up some bugs in oper_select_candidate(), notably the last loop which would return the *first * surviving-to-that-point candidate regardless of which one actually passed the test. This was producing such curious results as 'oid % 2' getting translated to 'int2(oid) % 2'
15	git-a3eb250	Fix the "close before dup" bug in clone-pack too Same issue as git-fetch-pack.	The same file
16	git-b3118bd	Fix incorrect error check while reading deflated pack data The loop in get_size_from_delta() feeds a deflated delta data from the pack stream _until_ we get inflated result of 20 bytes[*] or we reach the end of stream. Side note. This magic number 20 does not have anything to do with the size of the hash we use, but comes from 1a3b55c (reduce delta head inflated size, 2006-10-18).	In the same file
17	git-da0204d	Avoid scary errors about tagged trees/blobs during git-fetch This is the same bug as 42a32174b600f139b489341b1281fb1bfa14c252. The warning "Object $X is a tree, not a commit" is bogus and is not relevant here. If its not a commit we just need to make sure we don't mark it for merge as we fill out FETCH_HEAD.	Avoid scary errors about tagged trees/ blobs during git-fetch
18	git-cd03eeb	use write_str_in_full helper to avoid literal string lengths This is the same fix to use write_str_in_full() helper to write a constant string out without counting the length of it ourselves.	The same file

19	git-013aab	[PATCH] Dereference tag repeatedly until we get a non-tag.
		
		When we allow a tag object in place of a commit object, we only dereferenced the given tag once, which causes a tag that points at a tag that points at a commit to be rejected. Instead, dereference tag repeatedly until we get a non-tag.
		
		This patch makes change to two functions:
		
		- commit.c::lookup_commit_reference() is used by merge-base, rev-tree and rev-parse to convert user supplied SHA1 to that of a commit.
		- rev-list uses its own get_commit_reference() to do the same.
		
		Dereferencing tags this way helps both of these uses.
		
		if (obj->type == tag_type)
		- obj = ((struct tag *)obj)->tagged;
		
		if (object->type == tag_type) {
			The same commitment

20	linux-5bb1ab	http://git.kernel.org/?p=linux/kernel/git/torvalds/linux-2.6.git;a=commit;h=2570a4f5428bcdb10776223421817557 41e7fa60 ipv6: skb_dst() can be NULL in ipv6_hop_jumbo(). This fixes CERT-FI FICORA #341748 Discovered by Olli Jarva and Tuomo Untinen from the CROSS project at Codenomicon Ltd. Just like in CVE-2007-4567, we can't rely upon skb_dst() being non-NULL at this point. We fixed that in commit e76b2b2567b83448c2ee85a896433b96150c92e6 ("[IPV6]: Do no rely on skb->dst before it is assigned.")	http://git.kernel.org/?p=linux/kernel/git/torvalds/linux-2.6.git;a=commitdiff;h=483a47d2fe794328d29950fe00ce26dd405d9437;hp=3bd653c8455bc7991bae77968702b31c8f5df883

		However commit 483a47d2fe794328d29950fe00ce26dd405d9437 ("ipv6: added net argument to IP6_INC_STATS_BH") put a new version of the same bug into this function. Complicating analysis further, this bug can only trigger when network namespaces are enabled in the build. When namespaces are turned off, the dev_net() does not evaluate it's argument, so the dereference would not occur. So, for a long time, namespaces couldn't be turned on unless SYSFS was disabled. Therefore, this code has largely been disabled except by people turning it on explicitly for namespace development. With help from Eugene Teo eugene@redhat.com	ipv6: added net argument to IP6_INC_STATS_BH
21	linux-590929f	http://git.kernel.org/?p=linux/kernel/git/torvalds/linux-2.6.git;a=commitdiff;h=32127363eebdf63be2f375ed94838a4cdb1d6fe0;hp=590929f32adc3aaa702c287b624a0d0382730088 The implementation of the gain calculation for this sensor is incorrect. It is only working for the first 127 values. The reason is, that the gain cannot be set directly by writing a value into the gain registers of the sensor. The gain register work this way (see datasheet page 24): bits 0 to 6 are called "initial gain". These are linear. But bits 7 and 8 ("analog multiplicative factors") and bits 9 and 10 ("digital multiplicative factors") work completely different: Each of these bits increase the gain by the factor 2. So if the bits 7-10 are 0011, 0110, 1100 or 0101 for example, the gain from bits 0-6 is multiplied by 4. The order of the bits 7-10 is not important for the resulting gain. (But there are some recommended values for low noise) The current driver doesn't do this correctly: If the current gain is 000 0111 1111 (127) and the gain is increased by 1, you would expect the image to become brighter. But the image is completly dark, because the new gain is 000 1000 0000 (128). This means: Initial gain of 0, multiplied by 2. The result is 0. This patch adds a new function which does the gain calculation and also fixes the same bug for red_balance and blue_balance. Additionally, the driver follows the recommendation from the datasheet, which says, that the gain should always be above 0x0020.	Same commitment, same file
22	linux-9378b63	http://git.kernel.org/?p=linux/kernel/git/torvalds/linux-2.6.git;a=commitdiff;h=62627bec8a601c5679bf3d20a2096a1206d61b71;hp=9378b63ccb32b9c071dab155c96357ad1e52a709 x86: tsc: Fix calibration refinement conditionals to avoid divide by zero Konrad Wilk reported that the new delayed calibration crashes with a divide by zero on Xen. The reason is that Xen sets the pmtimer address, but reading from it returns 0xffffff. That results in the ref_start and ref_stop value being the same, so the delta is zero which causes the divide by zero later in the calculation. The conditional (!hpet && !ref_start && !ref_stop) which sanity checks the calibration reference values doesn't really make sense. If the refs are null, but hpet is on, we still want to break out. The div by zero would be possible to trigger by chance if both reads from the hardware provided the exact same value (due to hardware wrapping). So checking if both the ref values are the same should handle if we don't have hardware (both null) or if they are the same value (either by invalid hardware, or by chance), avoiding the div by zero issue. [tglx: Applied the same fix to native_calibrate_tsc() where this check was copied from]	Same commitment, same file

23	linux-fe1cbab	http://git.kernel.org/?p=linux/kernel/git/torvalds/linux-2.6.git;a=commitdiff;h=e75762fdcd27c1d0293d9160b3ac6dcb3371272a;hp=fe1cbabaea5e99a93bafe12fbf1b3b9cc71b610a Teach 9p filesystem to work in container with non-default network namespace. (Note: I also patched the unix domain socket code but don't have a test case for that. It's the same fix, I just don't have a server for it...) To test, run diod server (http://code.google.com/p/diod): diod -n -f -L stderr -l 172.23.255.1:9999 -c /dev/null -e /root and then mount like so: mount -t 9p -o port=9999,aname=/root,version=9p2000.L 172.23.255.1 /mnt	Same commitment
24	linux-d89197c	http://git.kernel.org/?p=linux/kernel/git/torvalds/linux-2.6.git;a=commitdiff;h=21fdc87248d1d28492c775e05fa92b3c8c7bc8db;hp=333ba7325213f0a09dfa5ceeddb056d6ad74b3b5 ath9k: fix two more bugs in tx power This is the same fix as commit 841051602e3fa18ea468fe5a177aa92b6eb44b56 Author: Matteo Croce <technoboy85@gmail.com> Date: Fri Dec 3 02:25:08 2010 +0100 The ath9k driver subtracts 3 dBm to the txpower as with two radios the signal power is doubled. The resulting value is assigned in an u16 which overflows and makes the card work at full power. in two more places. I grepped the ath tree and didn't find any others. scaledPower -= REDUCE_SCALED_POWER_BY_TWO_CHAIN;	http://git.kernel.org/?p=linux/kernel/git/torvalds/linux-2.6.git;a=commitdiff;h=841051602e3fa18ea468fe5a177aa92b6eb44b56;hp=d89197c7f34934fbb0f96d938a0d6cfe0b8bcb1c ath9k: fix bug in tx power The ath9k driver subtracts 3 dBm to t he txpower as with two radios the signal power is doubled. The resulting value is assigned in an u 16 which overflows and makes the card work at full power. scaledPower - = REDUCE_SCALED_POWER_BY _TWO_CHAIN;

25	linux7-cab758e	http://git.kernel.org/?p=linux/kernel/git/torvalds/linux-2.6.git;a=commitdiff;h=1eddceadb0d6441cd39b2c38705a8f5fec86e770;hp=cab758ef30e0e40f783627abc4b66d1b48fecd49
		
		Le jeudi 16 juin 2011 à 23:38 -0400, David Miller a écrit :
		> From: Ben Hutchings <bhutchings@solarflare.com>
		> Date: Fri, 17 Jun 2011 00:50:46 +0100
		>
		> > On Wed, 2011-06-15 at 04:15 +0200, Eric Dumazet wrote:
		> >> @@ -1594,6 +1594,7 @@ int tcp_v4_do_rcv(struct sock *sk, struct sk_buff *skb)
		> >> goto discard;
		> >>
		> >> if (nsk != sk) {
		> >> + sock_rps_save_rxhash(nsk, skb->rxhash);
		> >> if (tcp_child_process(sk, nsk, skb)) {
		> >> rsk = nsk;
		> >> goto reset;
		> >>
		> >
		> > I haven't tried this, but it looks reasonable to me.
		> >
		> > What about IPv6? The logic in tcp_v6_do_rcv() looks very similar.
		>
		> Indeed ipv6 side needs the same fix.
		>
		> Eric please add that part and resubmit. And in fact I might stick
		> this into net-2.6 instead of net-next-2.6
		>
		
		OK, here is the net-2.6 based one then, thanks !
		
		[PATCH v2] net: rfs: enable RFS before first data packet is received
		
		First packet received on a passive tcp flow is not correctly RFS
		steered.
			The same commitment

		One sock_rps_record_flow() call is missing in inet_accept() But before that, we also must record rxhash when child socket is setup	
26	linux-0029227	http://git.kernel.org/?p=linux/kernel/git/torvalds/linux-2.6.git;a=commitdiff;h=40a9fb17f32dbe54de3d636142a59288544deed7;hp=0029227f1bc30b6c809ae751f9e7af6cef900997 xhci: Do not run xhci_cleanup_msix with irq disabled	The same commitment
27	linux-713b3c9	http://git.kernel.org/?p=linux/kernel/git/torvalds/linux-2.6.git;a=commitdiff;h=4c7e604babd15db9dca3b07de167a0f93fe23bf4;hp=713b3c9e4c1a6da6b45da6474ed554ed0a48de69 ixgbe: fix panic due to uninitialised pointer Systems containing an 82599EB and running a backported driver from upstream were panicing on boot. It turns out hw->mac.ops.setup_sfp is only set for 82599, so one should check to be sure that pointer is set before continuing in ixgbe_sfp_config_module_task. I verified by inspection that the upstream driver has the same issue and also added a check before the call in ixgbe_sfp_link_config.	Same commitment
28	linux-52534f2	http://git.kernel.org/?p=linux/kernel/git/torvalds/linux-2.6.git;a=commitdiff;h=100f2341e305f98de3aa12fb472771ab029cbda7;hp=52534f2dba5d033c0c33e515faa2767d7e8e986a mtd: fix hang-up in cfi erase and read contention cfi erase command hangs up when erase and read contention occurs. If read runs at the same address as erase operation, read issues Erase-Suspend via get_chip() and the erase goes into sleep in wait queue. But in this case, read operation exits by time-out without waking it up. I think the other variants (0001, 0020 and lpddr) have the same problem too. Tested and verified the patch only on CFI-0002 flash, though.	Same commitment

29	linux-dcace06	http://git.kernel.org/?p=linux/kernel/git/torvalds/linux-2.6.git;a=commitdiff;h=6e83e10d92e12fa0181766a1fbb00d857bfab779;hp=1d56c453b14854637567c838109127b8decbf328
		
		mmc: dw_mmc: protect a sequence of request and request-done.
		
		Response timeout (RTO), Response crc error (RCRC) and Response error (RE) signals come with command done (CD) and can be raised preceding command done (CD). That is these error interrupts and CD can be handled in separate dw_mci_interrupt(). If mmc_request_done() is called because of a response timeout before command done has occured, we might send the next request before the CD of current request is finished. This can bring about a broken sequence of request and request-done.
		
		And Data error interrupt (DRTO, DCRC, SBE, EBE) and data transfer over (DTO) have the same problem.
		
		host->cmd_status = status;
		smp_wmb();
		set_bit(EVENT_CMD_COMPLETE, &host->pending_events);
		- tasklet_schedule(&host->tasklet);
			Same commitment

30	linux- a57ca04	mtd: jedec_probe: fix NEC uPD29F064115 detection linux v2.6.31-rc6 can not detect NEC uPD29F064115. uPD29F064115 is a 16 bit device. datasheet: http://www.cn.necel.com/memory/cn/download/M16062EJ2V0DS00.pdf This applies the same fix as used for SST chips in commit	The unlock_addr rework in kernel 2.6 .25 breaks 16-bit SST chips. SST 39LF160 and SST 39VF1601 are both 16-bit only chip (do not have BYTE# pin) and new uaddr value is not correc t for them. Add MTD_UADDR_0xAAAA_0x5555 fo r those chips. Tested with SST 39VF 1601

		ca6f12c67ed19718cf37d0f531af9438de85b70c ("jedec_probe: Fix SST 16-bit chip detection").	chip.

31	linux-ff0ac74	This is the same fix as commit 7959ea254ed18faee41160b1c50b3c9664735967 ("bnx2: Fix the behavior of ethtool when ONBOOT=no"), but for bnx2x:
		
		--- 	
		When configure in ifcfg-eth* is ONBOOT=no,
		the behavior of ethtool command is wrong.
		
		# grep ONBOOT /etc/sysconfig/network-scripts/ifcfg-eth2
		ONBOOT=no
		# ethtool eth2 | tail -n1
		Link detected: yes
		
		I think "Link detected" should be "no".
		--- 	
			I found a little bug.
			
			When configure in ifcfg-eth* is ONBOOT=no,
			the behavior of ethtool command is wrong.
			
			# grep ONBOOT /etc/sysconfig/network-scripts/ifcfg-eth2
			ONBOOT=no
			# ethtool eth2 | tail -n1
			Link detected: yes
			
			I think "Link detected" should be "no".

32	linux- 5153f7	Chuck Ebbert noticed that the desc_empty macro is incorrect. Fix it. Thankfully, this is not used as a security check, but it can falsely overwrite TLS segments with carefully chosen base / limits. I do not believe this is an issue in practice, but it is a kernel bug.	The same commitment
33	linux-8bea867	drivers/gpu/drm/drm_fb_helper.c: don't use private implementation of atoi() Kernel has simple_strtol() which would be used as atoi(). This is quite the same fix as in 2cb96f86628d6e97fcbda5fe4d8d74876239834c ("fbdev: drop custom atoi from drivers/video/modedb.c") because code in drivers/gpu/drm/drm_fb_helper.c is based on drivers/video/modedb.c.	fbdev: drop custom atoi from drivers/video/modedb.c Kernel has simple_strtol() implementation which could be used as atoi().
34	linux-ea2d8b5	iwl3945: fix deadlock on suspend This patch fixes iwl3945 deadlock during suspend by moving notify_mac out of iwl3945 mutex. This is a portion of the same fix for iwlwifi by Tomas.	iwlwifi: fix suspend to RAM in iwlwifi This patch fixes suspend to RAM after by moving notify_mac out of iwlwifi mutex
35	linux-c9a2c46	hwmon: (lm78) Fix I/O resource conflict with PNP Only request I/O ports 0x295-0x296 instead of the full I/O address range. This solves a conflict with PNP resources on a few motherboards. Also request the I/O ports in two parts (4 low ports, 4 high ports) during device detection, otherwise the PNP resource make the request (and thus the detection) fail. This is the exact same fix that was applied to driver w83781d in March 2008 to address the same problem: http://git.kernel.org/?p=linux/kernel/git/torvalds/linux-2.6.git;a=commit;h=2961cb22ef02850d90e7a12c28a14d74e327df8d	hwmon: (w83781d) Fix I/O resource conflict with PNP drivers/hwmon/w83781d.c
36	linux-d555009	USB: serial: fix race between unthrottle and completion handler in visor usb:usbserial:visor: fix race between unthrottle and completion handler visor_unthrottle() mustn't resubmit the URB unconditionally as the URB may still be running. the same bug as opticon.	USB: serial: fix race between unthrottle and completion handler in opticon
37	linux-9601e3f	Btrfs: fix fallocate deadlock on inode extent lock The btrfs fallocate call takes an extent lock on the entire range being fallocated, and then runs through insert_reserved_extent on each extent as they are allocated. The problem with this is that btrfs_drop_extents may decide to try and take the same extent lock fallocate was already holding. The solution	The same commitment

		used here is to push down knowledge of the range that is already locked going into btrfs_drop_extents. It turns out that at least one other caller had the same bug.	
38	linux-2567d71	rcu classic: new algorithm for callbacks-processing(v2) This is v2, it's a little deference from v1 that I had send to lkml. use ACCESS_ONCE use rcu_batch_after/rcu_batch_before for batch # comparison. rcutorture test result: (hotplugs: do cpu-online/offline once per second)	The same file different functions
39	linux-3976ae6	rt2x00: Only disable beaconing just before beacon update We should not write 0 to the beacon sync register during config_intf() since that will clear out the beacon interval and forces the beacon to be send out at the lowest interval. (reported by Mattias Nissler). The side effect of the same bug was that while working with multiple virtual AP interfaces a change for any of those interfaces would disable beaconing untill an beacon update was provided. This is resolved by only updating the TSF_SYNC value during config_intf(). In update_beacon() we disable beaconing temporarily to prevent fake beacons to be transmitted. Finally kick_tx_queue() will enable beaconing again.	hwmon: (w83627ehf) don't assume ba nk 0
40	linux-c09c518	hwmon: (w83627hf) don't assume bank 0 The bank switching code assumes that the bank selector is set to 0 when the driver is loaded. This might not be the case. This is exactly the same bug as was fixed in the w83627ehf driver two months ago: http://git.kernel.org/?p=linux/kernel/git/torvalds/linux- 2.6.git;a=commit;h=0956895aa6f8dc6a33210967252fd7787652537d In practice, this bug was causing the sensor thermal types to be improperly reported for my W83627THF the first time I was loading the w83627hf driver. From the driver history, I'd say that it has been broken since September 2005 (when we stopped resetting the chip by default at driver load.)	
41	linux-b45bfcc	IB/mlx4: Take sizeof the correct pointer in call to memset() When clearing the ib_ah_attr parameter in to_ib_ah_attr(), use sizeof *ib_ah_attr instead of sizeof *path. This is the same bug as was fixed for mthca in 99d4f22e ("IB/mthca: Use correct structure size in call to memset()"), but the code was cut and pasted into mlx4 before the fix was merged.	IB/mthca: Use correct structure size i n call to memset()
42	linux-34cc560	[TCP]: Prevent pseudo garbage in SYN's advertized window TCP may advertize up to 16-bits window in SYN packets (no window scaling allowed). At the same time, TCP may have rcv_wnd (32-bits) that does not fit to 16-bits without window scaling resulting in pseudo garbage into advertized window from the low-order bits of rcv_wnd. This can happen at least when mss <= (1<<wscale) (see tcp_select_initial_window). This patch fixes the handling of SYN advertized windows (compile tested only). …	[tcp_make_synack() has the same bu g, and I've added a fix for that to this patch -DaveM]
43	linux-efbfe96c	[PATCH] vmscan: Fix temp_priority race The temp_priority field in zone is racy, as we can walk through a reclaim path, and just before we copy it into prev_priority, it can be overwritten	The same commitment

		(say with DEF_PRIORITY) by another reclaimer. The same bug is contained in both try_to_free_pages and balance_pgdat, but it is fixed slightly differently. In balance_pgdat, we keep a separate priority record per zone in a local array. In try_to_free_pages there is no need to do this, as the priority level is the same for all zones that we reclaim from. …	
44	linux-093beac	IB/mthca: Fix posting lists of 256 receive requests to SRQ for Tavor If we post a list of length exactly a multiple of 256, nreq in doorbell gets set to 256 which is wrong: it should be encoded by 0. This is because we only zero it out on the next WR, which may not be there. The solution is to ring the doorbell after posting a WQE, not before posting the next one. This is the same bug that we just fixed for QPs with non-shared RQ.	Same commitment, but in the file drivers/infiniband/hw/mthca/mthca_qp.c
45	linux-a6230af	[CIFS] Fix cifs trying to write to f_ops patch 2ea55c01e0c5dfead8699484b0bae2a375b1f61c fixed CIFS clobbering the global fops structure for some per mount setting, by duplicating and having 2 fops structs. However the write to the fops was left behind, which is a NOP in practice (due to the fact that we KNOW the fops has that field set to NULL already due to the duplication). So remove it... In addition, another instance of the same bug was forgotten in november.	Same commitment
46	linux-c87e34e	[SCSI] sg: fix a bug in st_map_user_pages failure path sg's st_map_user_pages is modelled on an earlier version of st's sgl_map_user_pages, and has the same bug: if get_user_pages got some but not all of the pages, then those got were released, but the positive res code returned implied that they were still to be freed.	[SCSI] st: fix a bug in sgl_map_user_pages failure path
47	linux-5917583	[PATCH] mm: move_pte to remap ZERO_PAGE Move the ZERO_PAGE remapping complexity to the move_pte macro in asm-generic, have it conditionally depend on __HAVE_ARCH_MULTIPLE_ZERO_PAGE, which gets defined for MIPS. For architectures without __HAVE_ARCH_MULTIPLE_ZERO_PAGE, move_pte becomes a noop. From: Hugh Dickins <hugh@veritas.com> Fix nasty little bug we've missed in Nick's mremap move ZERO_PAGE patch. The "pte" at that point may be a swap entry or a pte_file entry: we must check pte_present before perhaps corrupting such an entry	Linux v2.6.14-rc2 Avast, ye scurvy land-lubbers! Time to try out a new release. Arrr!
48	linux-19147bb	e1000: fix unmap bug This is in reference to the issue shown in kerneloops (search e1000 unmap) The e1000 transmit code was calling pci_unmap_page on dma handles that it might have called pci_map_single on. Same bug as e1000e	e1000e: fix unmap bug This is in reference to https://bugzilla.redhat.com/show_bug.cgi?id=484494 Also addresses issue show in kerneloops The e1000e transmit code was calling pci_unmap_page on dma handles that it might have called pci_map_single on.
49	linux-4c25a2c	As we just did for context cache flushing, clean up the logic around whether we need to flush the iotlb or just the write-buffer, depending on caching mode. Fix the same bug in qi_flush_iotlb() that qi_flush_context() had -- it isn't supposed to be returning an error; it's supposed to be returning a flag which triggers a write-buffer flush.	The same commitment

		Remove some superfluous conditional write-buffer flushes which could never have happened because they weren't for non-present-to-present mapping changes anyway.	
50	linux-529ed80	These patch fix a longstanding bug in the i810 frame buffer driver. The handling of the i2c bus is wrong: A 1 bit should not written to the i2c, these will be done by switch the i2c to input. Driving an 1 bit active is against the i2c spec. An active driven of a 1 bit will result in very strange error, depending which side is the more powerful one. In my case it depends on the temperature of the Display-Controller-EEprom: With an cold eprom a got the correct EDID datas, with a warm one some of the 1 bits was 0 :-(The same bug is also in the intelfb driver in the file drivers/video/intelfb/intelfb_i2c.c. The functions intelfb_gpio_setscl() and intelfb_gpio_setsda() do drive the 1 bit active to the i2c bus. But since i have no card which is used by the intelfb driver i cannot fix it.	The same commitment
51	linux-3083e83	Same fix as f844a709a7d8f8be61a571afc31dfaca9e779621 "iwlwifi: do not set tx power when channel is changing"	Mac80211 can request for tx power a nd channel change in one ->config call. If that happens, *_send_tx_powe r functions will try to setup tx power for old channel, what can be no t correct because we already change the band. I.e error "Failed to get cha nnel info for channel 140 [0]", can be printed frequently when operat ing in software scanning mode.
52	linux-78794b2	Michael Leun reported that running parallel opens on a fuse filesystem can trigger a "kernel BUG at mm/truncate.c:475" Gurudas Pai reported the same bug on NFS. The reason is, unmap_mapping_range() is not prepared for more than one concurrent invocation per inode. For example: thread1: going through a big range, stops in the middle of a vma and stores the restart address in vm_truncate_count. thread2: comes in with a small (e.g. single page) unmap request on the same vma, somewhere before restart_address, finds that the vma was already unmapped up to the restart address and happily returns without doing anything. …	The same commitment
53	linux-c594d88	This fixes a race between the glock and the page lock encountered during truncate in gfs2_readpage and gfs2_prepare_write. The gfs2_readpages function doesn't need the same fix since it only uses a try lock anyway, so it will fail back to gfs2_readpage in the case of a potential deadlock. This bug was spotted by Russell Cattelan.	Same commitment
D1	git-d53fe81	archive: centralize archive entry writing Add the exported function write_archive_entries() to archive.c, which uses the new ability of read_tree_recursive() to pass a context pointer to its callback in order to centralize previously duplicated code. The new callback function write_archive_entry() does the work that every archiver backend needs to do: loading file contents, entering subdirectories, handling file attributes, constructing the full path of the entry. All that done, it calls the backend specific write_archive_entry_fn_t function.	
D2	git-3fe2a89	status: reduce duplicated setup code We have three output formats: short, porcelain, and long. The short and long formats respect user-config, and the porcelain one does not. This led to us repeating	

		config-related setup code for the short and long formats. Since the last commit, color config is explicitly cleared when showing the porcelain format. Let's do the same with relative-path configuration, which enables us to hoist the duplicated code from the switch statement in cmd_status. As a bonus, this fixes "commit --dry-run --porcelain", which was unconditionally setting up that configuration, anyway.	
D3	git-e923eae	refactor duplicated fill_mm() in checkout and merge-recursive The following function is duplicated: fill_mm Move it to xdiff-interface.c and rename it 'read_mmblob', as suggested by Junio C Hamano. Also, change parameters order for consistency with read_mmfile().	
D4	git-e923eae-2	connect.c: move duplicated code to a new function 'get_host_and_port' The following functions: git_tcp_connect_sock (IPV6 version) git_tcp_connect_sock (no IPV6 version), git_proxy_connect have common block of code. Move it to a new function 'get_host_and_port'	
D5	linux-23edcc4	tcp: Add tcp_validate_incoming & put duplicated code there Large block of code duplication removed. Sadly, the return value thing is a bit tricky here but it seems the most sensible way to return positive from validator on success rather than negative. net/ipv4/tcp_input.c 4904-4909 parents: orinoco: Add MIC on TX and check on RX Use the MIC algorithm from the crypto subsystem. 23edcc4147ad36f8d55f0eb79c21e245ffb9f211 52second generate pdg	
D6	linux-ec33679	fs: consolidate dentry kill sequence The tricky locking for disposing of a dentry is duplicated 3 times in the dcache (dput, pruning a dentry from the LRU, and pruning its ancestors). Consolidate them all into a single function dentry_kill. fs/dcache.c 304-310 parent: ec33679d78f9d653a44ddba10b5fb824c06330a1 fs: use RCU in shrink_dentry_list to reduce lock nesting 44second generate pdg	
D7	linux-2644487	drm/i915: overlay: extract some duplicated code I've suspected some bug there wrt to suspend, but that was not the case. Clean up the code anyway. drivers/gpu/drm/i915/intel_overlay.c 441-446 parents: drm/i915: remove Pineview EOS protection support	

		HW guys have an evaluation about the impact about EOS, and say the impact is quite small, so they have removed EOS detection support. This patch removes EOS feature. revert commit 043029655816ed4cfc2ed247020ef97e5d637392 directly reverting it gives a hunk error, so please use this one. 26444877812fb2a2b9301b0b3702fdf9f9e06e4b 121second generate pdg	
D8	linux-a4e77d0	With 2.6.27-rc3 I noticed the following messages in my boot log: 0000:01:00.0: 0000:01:00.0: Warning: detected DSPD enabled in EEPROM 0000:01:00.0: eth0: (PCI Express:2.5GB/s:Width x1) 00:16:76:04:ff:09 The second seems correct, but the first has a silly repetition of the PCI device before the actual message. The message originates from e1000_eeprom_checks in e1000e/netdev.c. With this patch below the first message becomes e1000e 0000:01:00.0: Warning: detected DSPD enabled in EEPROM which makes it similar to directly preceding messages. Use dev_warn instead of e_warn in e1000_eeprom_checks() as the interface name has not yet been assigned at that point. [akpm@linux-foundation.org: coding-style fixes] drivers/net/e1000e/netdev.c 4671-4674 parents: atl1e: remove the unneeded (struct atl1e_adapter *) Remove the unneeded (struct atl1e_adapter *) casts, for hw->adapter already has type atl1e_adapter *. a4e77d063d61e4703db813470fefe90dac672b55	