find.24e2271e
Error Type: Functional Bug
Avg. Time: 13.8 min
Explanation: Slightly difficult
Patching: Slightly difficult
Correctness: 75%

If find is set to print the found file's base directory followed by the found file's name (-printf '%H %P\n') and there exist directories of different length, then find incorrectly splits base directory and file name during printing. Because the index state.starting_path is set only for the first working directory (`ftsfind.c:278-279`) the incorrect value of state.starting_path is used when printing base directory and file name (`pred.c:709-718`, `pred.c:813`). **Examples of Correct Fixes**: 1) Recompute state.starting_path_length for each argv before calling find. 2) Weaken condition that prevents state.starting_path_length to be reset. **Example of Incorrect Fix**: Always update state.starting_path_length even if ent->fts_level != 0 (*Regression* because it then carries the incorrect "starting_path_length").

find.dbcb10e9
Error Type: Crash
Avg. Time: 22.9 min
Explanation: Slightly difficult
Patching: Slightly difficult
Correctness: 81%

If find is set to print all files that are exactly 2 days old (-mtime 2), it crashes with a segmentation fault. Variable **pend is defined as pointer pointer (`parser.c:2739`) and expected to be allocated when xstrtoumax is called (`parser.c:2759`). However, it is still NULL after the call such that the null pointer check for pend* is itself a null pointer dereference (`parser.c:2762`). **Examples of Correct Fixes**: 1) Add null pointer check for pend. 2) Change definition of **pend to *pend and update references. 3) Allocate memory for **pend. **Examples of Incorrect Fixes**: 1) Remove code containing null pointer dereference (*Treating the Symptom*). 2) Change the check involving **pend (*Treating the Symptom* because the nullpointer is still dereferenced, only the program does not crash).

find.07b941b1
Error Type: Crash
Avg. Time: 23.7 min
Explanation: Slightly difficult
Patching: Slightly difficult
Correctness: 80%

If find is set to search for file matching a regular expression (-regex '.*'), the argument pointer arg_ptr is incremented (`parser.c:1644`) before it is used (`parser.c:1645`) which results in a nullpointer dereference (`parser.c:926`). **Examples of Correct Fixes**: 1) Increment arg_ptr *after* argv[*arg_ptr] is read. 2) Save the previous value of argv[*arg_ptr] in a temporary variable and use this one. **Examples of Incorrect Fixes**: 1) Do not increment arg_ptr at all (*Regression* because other arguments may not be parsed at all). 2) Add null-pointer check (*Incomplete Fix* because estimate_pattern_match is still called with a nullpointer).

find.c8491c11
Error Type: Crash
Avg. Time: 31.4 min
Explanation: Slightly difficult
Patching: Slightly difficult
Correctness: 54%

If find is set to print files that are newer than a reference file and this reference file is not specified (-newerXY), find crashes with a segmentation fault. This is caused by incrementing the argument pointer arg_ptr without a bounds check (`parser.c:1315`) resulting in a null pointer dereference (`lib/quotearg.c:249`). **Examples of Correct Fixes**: Check for nullpointer directly after increment of arg_ptr. **Examples of Incorrect Fixes**: 1) Check for nullpointer only before or in fatal_file_error (*Incomplete Fix* because null pointer might still propagate via parser.c:1342 or parser.c:1347). 2) Do not increment the pointer at all (*Regression* because some arguments may not be parsed, at all).

find.6e4cecb6
Error Type: Functional Bug
Avg. Time: 38.2 min
Explanation: Moderately difficult
Patching: Not at all difficult
Correctness: 89%

If find is set to search a directory referenced by a symbolic link and containing a file, and find is set to follow symbolic links (-L) or to not follow symbolic links except for those set to be searched (-H), then find does not print the file in the referenced directory and instead reports "Too many levels of symbolic links". Because of a mixup in the condition of a ternary operator (`find.c:1094`), extraflags are set to O_NOFOLLOW when it should be 0 and to 0 when it should be O_NOFOLLOW. The flag controls whether symlinks are followed when a directory is opened (`find.c:1097`). Because of this fault, safely_chdir returns SafeChdirFailSymlink (`find.c:1618`) whence the error message is printed (`find.c:1642`). **Example of Correct Fix**: Fix ternary operator. **Example of Incorrect Fix**: Do not fail if safely_chdir returns SafeChdirFailSymlink (*Treating the Symptom*).

find.091557f6
Error Type: Crash
Avg. Time: 44.8 min
Explanation: Slightly difficult
Patching: Slightly difficult
Correctness: 54%

If find is set to search for files (-type f) while following symbolic links (-L) and a symbolic link loop exists, then it aborts with a coredump instead of listing the symbolic links and terminating gracefully. If a symbolic link loop exists, no stat information is available and the flag FTS_NS is set (`ftsfind:584`). The flag is not properly handled (`ftsfind.c:425-446`), such that state.type and mode are incorrectly set (`ftsfind.c:460`) and the assertion fails (`pred.c:1578`). **Example of Correct Fix**: Handle FTS_NS flag. **Examples of Incorrect Fixes**: 1) Remove violated assertion (*Treating the Symptom*). 2) Force stat() to be called such that stat information is available (*Incorrect Workaround* because stat() is not supposed to be called on symlink loops).

find.24bf33c0
Error Type: Crash
Avg. Time: 45.1 min
Explanation: Moderately difficult
Patching: Slightly difficult
Correctness: 50%

If find is set to search for files (-type f) while following symbolic links (-L) and a symbolic link loop exists, then it still prints the looping links while an error message is expected. If a symbolic link loop exists, no stat information is available and the flag FTS_NS is set (`ftsfind.c:586`). The flag is not properly handled (`ftsfind.c:431-446`) so that the links are printed (`pred.c:1459`). **Example of Correct Fix**: Handle FTS_NS as error IF symlink loop. **Examples of Incorrect Fixes**: 1) Handle FTS_NS as error independent of whether it is a symlink loop (*Regression* because FTS_NS alone does not indicate an error). 2) Handle all flags as error (*Regression* because not all flags indicate errors).

find.183115d0
Error Type: Resource Leak
Avg. Time: 49.2 min
Explanation: Slightly difficult
Patching: Slightly difficult
Correctness: 83%

If we ulimit the number file descriptors that can be open simulatanously and set find to execute ls for every subdirectory (-execdir ls '{}' \;), it quickly runs out of file descriptors. File descriptors are always opened (`pred.c:520`) but never closed (`pred.c:659-664`) which raises an error when no more descriptors are available (`pred.c:579`). **Example of Correct Fix**: Close file descriptor as soon as it is not used anymore. **Example of Incorrect Fix**: Close random file descriptor (*Incomplete Fix* because still leaking file descriptors).

find.93623752
Error Type: Functional Bug
Avg. Time: 50.8 min
Explanation: Moderately difficult
Patching: Slightly difficult
Correctness: 92%

There are two errors: 1) If find is set to search for files that were changed in the last n days but n is not a number (-ctime x), then find complains about a "missing" argument instead of reporting the "incorrect" argument. Function parse_time calls collect_args to assign the current argument argv[*arg_ptr] to timearg and increment the argument pointer arg_ptr (`parser.c:3102`). When timearg is failed to be parsed as a number, parse_time returns without decrementing arg_ptr (`parser.c:3127-3128`). When the error is reported (`tree.c:1248-1271`), the argument pointer points to NULL directly after the incorrect argument (`tree.c:1250`), such that the error is reported as missing argument instead of invalid argument. 2) If find is set to search for files belonging to a certain group but the group-id is not specified or not a number (-gid x), then find crashes with a segmentation fault. When the argument following the –gid option is being parsed (`parser.c:913`), insert_num returns NULL because argv[*arg_ptr] is NULL or not a number (`parser.c:3235-3259`). This nullpointer remains unchecked and is dereferenced leading to a segmentation fault (`parser.c:914`). When nullpointer dereference is fixed the same symptom is observed for -gid as for -ctime because the argument pointer is also forgot to be decremented. **Examples of Correct Fixes**: For first error, 1) decrement/restore arg_ptr when parsing of second argument of an option fails or 2) no copy of old argument during error-reporting. For second error, add null pointer check. **Example of Incorrect Fix**: For first error, decrement argument pointer before even calling parse_time (*Regression* because even correct arguments are reported as incorrect ones).

find.66c536bb
Error Type: Functional Bug
Avg. Time: 55.5 min
Explanation: Moderately difficult
Patching: Slightly difficult
Correctness: 92%

If find is set to print files that are strictly younger than 2 days (-mtime -2), it will instead print files that are exactly 2 days old. The function get_comp_type actually increments the argument pointer timearg (`parser.c:3175`). So, when the function is called the first time (`parser.c:3109`), timearg still points to '-'. However, when it is called the second time (`parser.c:3038`), timearg already points to '2' such that it is incorrectly classified as COMP_EQ (`parser.c:3178`). **Examples of Correct Fixes**: 1) Save timearg in auxiliary variable and restore after first call to get_comp_type. 2) Pass a copy of timearg into the first call of get_comp_type. 3) Pass a copy of timearg into get_relative_timestamp (which calls get_comp_type the second time). 4) Decrement timearg after the first call to get_comp_type. **Example of Incorrect Fix**: Restore timearg only if classified as COMP_LT (*Incomplete Fix* because it does not solve the problem for -mtime +2).

find.b445af98
Error Type: Functional Bug
Avg. Time: 56.5 min
Explanation: Moderately difficult
Patching: Slightly difficult
Correctness: 50%

If find is set to search a directory containing a symbolic link, to not follow any symbolic links (except for those specified on the command line; -H), and to print only symbolic links (-type l), then find does not print the link. The root cause is that state.cur_depth is used before it is set. When digest_mode checks whether to follow symlinks (`util.c:629`), state.curdepth is still 0 (`util.c:607`), so that mode are incorrectly set to follow symlinks (`util.c:630-636`). Only later state.curdepth is set (`ftsfind.c:230`). Because of the incorrect value of mode, it is incorrectly decided not to print the file (`pred.c:1749`). **Example of Correct Fix**: Move state.curdepth assignment to shortly before digest_mode is called. **Examples of Incorrect Fixes**: 1) Change check to match incorrect value (0) of state.curdepth (*Treating the Symptom*). 2) Force stat() to be called such that stat information is available (*Incorrect Workaround* because stat() is not supposed to be called on symlink loops).

find.ff248a20
Error Type: Infinite Loop
Avg. Time: 57.7 min
Explanation: Moderately difficult
Patching: Moderately difficult
Correctness: 40%

If find is set to search a directory containing a symbolic link that references an ancistor directory and if find is set to follow symlinks (-follow), then it runs indefinitely. The global variable dir_ids tracks the directories that have already been visited. The function process_path would correctly exit with a loop warning (`find.c:1428-1434`) if the current directory (in stat_buf) has already been visited. However, after the current directory is correctly added to those that have already been visited (`find.c:1442`), the same entry is overriden with uninitialized values (`find.h:1621`) such that the current directory is never marked as already visited. **Examples of Correct Fixes**: 1) Remember whether stat() has been called. If not done, call stat() before overriding dir_ids[dir_curr] at find/find.c:1621. 2) Always stat() before overriding dir_ids[dir_curr] at find/find.c:1621 such that statbuf is initialized. 3) Only overwrite dir_ids[dir_curr] if statbuf is initialized. **Examples of Incorrect Fixes**: 1) Never override dir_ids[dir_curr] (*Regression* because it isn't overridden when it should be). 2) Follow links to a maximum depth of 1 (*Regression* because symlinks might need to be followed to an arbitrary depth).

find.c6680237
Error Type: Functional Bug
Avg. Time: 76.4 min
Explanation: Moderately difficult
Patching: Moderately difficult
Correctness: 27%

If find is set to search a directory containing three other directories which contain the folder "bug" and to execute pwd in every folder containing the folder "bug" (-name bug -execdir pwd \;), then find prints the first directory three times. The reason is that the working directory specified in execp->wd_for_exec is set only once (`pred.c:513-527`) and never updated. **Examples of Correct Fixes**: 1) Correct buggy if-condition by substituting execp->wd_for_exec by execp->todo. 2) If is_exec_in_local_dir, then always reallocate execp->wd_for_exec and remove the assertion. **Example of Incorrect Fix**: Remove if-condition such that it always redefines execp->wd_for_exec and keep assertion that execp->todo is false (*Regression* because execp->todo might be true such that assertion may fail).

find.e1d0a991
Error Type: Functional Bug
Avg. Time: 88.2 min
Explanation: Very difficult
Patching: Very difficult
Correctness: 17%

If find is set to a directory containing a file, to follow symbolic links (-L), and to execute ls for every subdirectory (-execdir ls '{}' \;), then find incorrectly also prints the base directory. If find is set to follow symlinks, the flag FTS_LOGICAL is set (`ftsfind.c:349`) before the directory search is initiated (`ftsfind.c:364`). When a directory is searched (`ftsfind.c:373`), the working directory is not changed because FTS_LOGICAL is set. Hence, the *full* pathname is passed as argument to execdir (`pred.c:484-490` and `pred.c:467-471`). **Example of Correct Fix**: Correctly compute pathname and prefix in new_impl_pred_exec. **Example of Incorrect Fix**: Remove FTS_LOGICAL flag (Incorrect Workaround because FTS_LOGICAL is supposed to be set).

grep.55cf7b6a
Error Type: Functional Bug
Avg. Time: 21.1 min
Explanation: Slightly difficult
Patching: Not at all difficult
Correctness: 91%

If grep is set to silently skip devices, FIFOs, and sockets (-D skip), then grep does not search on standard input when no file is provided. When the skip option is enabled, variables devices is set to SKIP_DEVICES (`main.c:1852-1859`). If no file is provided, variable file is NULL and variable desc is set to STDIN_FILENO (`main.c:1217-1218`). The code which handles SKIP_DEVICES (`main.c:1246-1255`) decides to skip STDIN (which is a special device) even though it should not (desc == STDIN_FILENO). **Examples of Correct Fixes**: 1) Do not skip if desc is set to STDIN_FILENO. 2) Do not skip if file is not set (and thus desc is set to STDIN_FILENO). **Example of Incorrect Fix**: Negate the skip condition (*Regression* because it skips everything that should not be skipped while indeed not skipping STDIN).

grep.54d55bba
Error Type: Crash
Avg. Time: 26.7 min
Explanation: Slightly difficult
Patching: Slightly difficult
Correctness: 69%

If grep is set to search in all files under each directory recursively (-r) but to exclude certain directories (–exclude-dir=foo), then grep crashes with a segmentation fault. When grepdir computes the name_space (`src/grep.c:1361`), it calls function isdir1 via function savedir (`lib/savedir.c:123`). Now, the code in isdir1 that is supposed to remove the trailing slashes from the directory name uses the uninitialized variable path instead of variable dir (`lib/savedir.c:51`). The nullpointer dereference results in a segmentation fault. **Example of Correct Fix**: Substitute path with dir. **Examples of Incorrect Fixes**: 1) Return if path is not initialized (*Regression* because isdir1 returns false even if dir is a directory). 2) Only use path if initialized (*Regression* because isdir1 does not remove trailing slashes).

grep.9c45c193
Error Type: Functional Bug
Avg. Time: 37.7 min
Explanation: Moderately difficult
Patching: Slightly difficult
Correctness: 83%

If grep is set to search only specific files (–include=a.txt), then grep does not print a match even if there is one. First, main correctly adds the include pattern with EXCLUDE_INCLUDE flag set (`grep.c:2136-2140`). When the files are chosen for the search, files that are supposed to be included are actually excluded because the return value of excluded_file_name is unnecessarily negated (`grep.c:2267-2269`). The negation is unnecessary because the function exclude_file_name is incorrectly assumed to treat excludes and includes the same. However, the behavior changes if the EXCLUDE_INCLUDE flag is present (`lib/exclude.c:410`, `lib/exclude.c:359`). **Examples of Correct Fixes**: 1) Remove negation such that included_patterns are not excluded during classification. 2) Do not set EXCLUDE_INCLUDE flag for included_patterns which effectively negates the faulty condition. **Example of Incorrect Fix**: Independent of whether a file matches the included_pattern, never exclude (*Regression* because it doesn't skip files that are *not* in the included patterns).

grep.5fa8c7c9
Error Type: Infinite Loop
Avg. Time: 38.8 min
Explanation: Moderately difficult
Patching: Slightly difficult
Correctness: 50%

If grep is set to search for fixed strings (-F), the empty string is given (""), and the locale is UTF8, then grep runs undefinitely. When FExecute searches for a match of the empty string, variable len contains the size of the match; here, len=0 (`kwsearch.c:106`). Because len=0, the check is_mb_middle (`search utils.c:117-146`) whether the match occurs within a multibyte character returns true (`kwsearch.c:108`). However, the size of the supposed multibyte character is computed as mb_len=1 (`kwsearch.c:115`). When mb_len-1 is added to beg (`kwsearch.c:118`) to advance behind the supposed multibyte character, beg's value remains unchanged. The loop is continue'd (`kwsearch.c:121`). Since beg has the same value every time the loop exit condition is checked (`kwsearch.c:101`), the loop exit condition never holds, resulting in an infinite loop. **Examples of Correct Fixes**: 1) Function is_mb_middle returns false for len=0. 2) Only call is_mb_middle if len is set. 3) Jump to success if mb_len==1. **Examples of Incorrect Fixes**: 1) Remove continue (*Treating the Symptom*). 2) Don't reset beg (*Regression* because it breaks multibyte character handling). 3) Remove part of the check which causes is_mb_middle to return true (*Regression* because it breaks multibyte character handling). 4) Do not compute match_size but return complete buffer until end of line (*Regression* because only match should be returned).

grep.db9d6340
Error Type: Infinite Loop
Avg. Time: 40.6 min
Explanation: Slightly difficult
Patching: Slightly difficult
Correctness: 45%

If grep conducts a fixed-strings search (-F) for a pattern that contains multibyte characters, then it runs indefinitely. When EXECUTE_FCT finds a match in the middle of a multibyte character, it is supposed to continue after the multibyte character (`search.c:638-639`). However, the beginning of the next multibyte character is not found, and mb_start remains unchanged (`search.c:228-256`). After beg is assigned mb_start minus 1, the loop is continue'd (`search.c:640`). The loop exit condition never holds (`search.c:632`) because beg never exceeds buf + size, resulting in an infinite loop. **Examples of Correct Fixes**: 1) Raise an error, if is_mb_middle is unsuccessful in finding the beginning of the multi-byte and adjusting mb_start. 2) Go to after the current match. **Examples of Incorrect Fixes**: 1) Remove continue (*Treating the Symptom*). 2) Do not reset beg (*Regression* because it breaks multibyte character handling).

grep.2bc0c659
Error Type: Functional Bug
Avg. Time: 47.2 min
Explanation: Moderately difficult
Patching: Moderately difficult
Correctness: 13%

If grep conducts a case-insensitive search (-i) in a file containing 8-bit characters and the current locale is Turkish UTF8, then grep prints the wrong output. When grep conducts a case-insensitive search, it lowers the case of the input string before matching (`search.c:384-392`). The lower case of an upper-case 8-bit character might occupy one more or less bytes. The latter case is not handled. When the match_size is computed (`grep.c:1081`), the lower-case match is used (`grep.c:1060-1062`). When the match is printed, the incorrect lower-case match size which is usually larger than the actual match size is used (`grep.c:1085-1091`). **Examples of Correct Fixes**: 1) Update the map that maps lower-case character to the normal case characters to account for cases where the number of bytes it occupies *decreases* in the lower-case. 2) To correct the match_size, lower-case as many characters in the normal-case match as result in match_size lower-case characters. **Examples of Incorrect Fixes**: 1) Return complete line if match exists (*Regression* because only the match should be returned). 2) Add the difference in length of lower-case and normal-case string to the match size (*Incomplete Fix* because for files that have more multibyte characters than given in the match, grep reports longer matches than needed).

grep.8f08d8e2
Error Type: Functional Bug
Avg. Time: 48.4 min
Explanation: Moderately difficult
Patching: Moderately difficult
Correctness: 75%

If grep is set to search for lines containing whole words that match a regular expression (-w), it prints only the match instead of the complete line. When execute searches for a match, it correctly sets variable len to the length of the match (`search.c:388`). When it is checked if the match aligns with word boundaries (`search.c:408-414`), the match length len still points to the end of the match. So, execute returns the length of the match instead of the end of the line (`grep.c:997`). **Examples of Correct Fixes**: 1) Add statement: goto success (which updates len with end - beg). 2) Update len with end - beg. **Example of Incorrect Fix**: Always return complete line (*Regression* because in some settings grep should return only the match).

grep.58195fab
Error Type: Functional Bug
Avg. Time: 50.5 min
Explanation: Moderately difficult
Patching: Slightly difficult
Correctness: 82%

If grep is set to search all TXT files (–include="*.txt") but excluding some files (–exclude="foo.txt"), then grep also searches files that are not TXT ignoring the include option. Because included_patterns is not initialized with EXCLUDE_WILDCARDS (`src/grep.c:2137`), the exclude pattern is not added in add_exclude (`lib/exclude.c:449`). Files are matched exactly (treating "*.txt" as file name) instead of using wildcards (`lib/exclude.c:417-427`). These files are then incorrectly classified as included/excluded (`src/grep.c:2261-2271`). **Examples of Correct Fixes**: 1) Add EXCLUDE_WILDCARDS flag for includes. 2) Add EXCLUDE_INCLUDE flags for excludes if there are includes. **Examples of Incorrect Fixes**: 1) Substitute EXCLUDE_INCLUDE with EXCLUDE_WILDCARDS for includes (*Regression* because EXCLUDE_INCLUDE flags must also be set for includes). 2) Negate condition that decides whether to exclude (*Regression* because files that are specified to be excluded are now included).

grep.c1cb19fe
Error Type: Functional Bug
Avg. Time: 58.4 min
Explanation: Very difficult
Patching: Slightly difficult
Correctness: 71%

If grep searches for string specified in a bracket expression, then for some UTF8 locales (ru_RU.UTF-8) grep does not print a match. For some locales dfaparse sets the global flag hard_LC_COLLATE (`dfa.c:1418`) to denote that characters are ordered in a strange way (e.g. Russian cyrilic). If hard_LC_COLLATE is set, then lex prepares the info about the letters in the bracket expression and finally calls in_coll_range (`dfa.c:1103-1116`). Now, in_coll_range uses the correct function strcoll to compare the letters, but the condition is incorrect and the wrong character are selected to be in the range that is specified by the bracket expression. Hence, there is no match reported. **Example of Correct Fix**: Fix the simple operator fault. **Examples of Incorrect Fixes**: 1) Fix locale, such that multibyte characters do not need to be handled (*Regression* because LC_ALL is supposed to be handled). 2) Implement in_coll_range as locale implemented match (*Regression* because match is supposed to be locale dependent).

grep.7aa698d3
Error Type: Functional Bug
Avg. Time: 59.9 min
Explanation: Moderately difficult
Patching: Moderately difficult
Correctness: 13%

If grep conducts a case-insensitive search (-i) on an input that contains multibyte characters and the locale is UTF8, then grep prints a match of incorrect length. When conducting the case-insensitive search, EXECUTE_FCT first computes a lower-case of the input (`search.c:388`). The length of the match is computed for the match in the lower-case input (`search.c:555`). However, the lower-case of a multibyte character can take 1 byte less. So, the length of the normal-case and lower-case input differ. The computed value of match_size could be half the expected value (`grep.c:1081-1085`). Hence, the match in the normal-case input is printed with incorrect length (`grep.c:1091`). **Example of Correct Fix**: Add a mapping between normal-case and lower-case string to compute the length of the match in the normal-case string from the length of the match in the lower-case string. **Examples of Incorrect Fixes**: 1) Do not lower the case (*Regression* because a case-insensitive search is case-sensitive). 2) If matched string contains a multibyte char, double the match_size (*Incomplete Fix* because it works only of all are multibyte characters). 3) Print complete line if there is a match (*Regression* because only match should be returned).

grep.3220317a
Error Type: Crash
Avg. Time: 63.7 min
Explanation: Moderately difficult
Patching: Moderately difficult
Correctness: 20%

If grep searches for a bracket expression containing a multibyte character in a file that contains multibyte characters and the current locale is UTF8, then grep crashes with a segmentation fault. When parse_bracket_exp parses the next character, array index c is assigned EOF (-1) if the character is multibyte (`dfa.c:498`, `dfa.c:363`) while wc is assigned the correct index. However, when parse_bracket_exp calls setbit_case_fold (`dfa.c:697`) it uses c which overflows during the cast from int to unsigned. After setbit_case_fold has called setbit (`dfa.c:274`), the array is accessed at a too large index which causes a segmentation fault (`dfa.c:168`). **Example of Correct Fix**: Use wc instead of c (which equals c if the character is not multibyte). **Examples of Incorrect Fixes**: 1) Check for overflow condition c=EOF (*Treating the Symptom* because multibyte characters are still handled incorrectly). 2) Use an arbitrary value instead of c (*Treating the Symptom* because while it does not crash, the bracket expression is not correctly handled).

grep.3c3bdace
Error Type: Crash
Avg. Time: 64.8 min
Explanation: Very difficult
Patching: Moderately difficult
Correctness: 70%

If grep searches for a certain extended regular expression (-E '(.)*(\1)'), then it crashes with a coredump. When dfaanalyze allocates memory for merged.elems (`dfa.c:1728`), it allocates insufficient memory because merged.elems can grow to twice the original size (`dfa.c:1455`). Then memory is corrupted when the array is accessed out of bounds (`dfa.c:1453`). Only later the program crashes because of the corrupted memory (`dfa.c:1917`). **Examples of Correct Fixes**: 1) Allocate twice or 3x as much for merged.elems. 2) Reallocate as needed. **Example of Incorrect Fix**: Always reset the number of elements (nelem) to 0 (*Regression* because we always override the first element).

grep.c96b0f2c
Error Type: Functional Bug
Avg. Time: 67.6 min
Explanation: Very difficult
Patching: Moderately difficult
Correctness: 50%

If grep conducts a case-insensitive search (-i) for the empty line (""$") and an UTF-8 locale is set, then grep prints matches even for non-empty lines. For case-sensitive searches or 8-bit locales, execute is called with the complete buffer and correctly returns no match (`grep.c:1045-1046`). Otherwise, execute is called for each line (`grep.c:1048-1063`). However, execute does not handle the case when no match is found (`search.c:388`), which is why the non-match is printed (`grep.c:1091`). **Examples of Correct Fixes**: 1) Handle case where no match was found by breaking loop if next_beg == buflim. 2) Skip printing if match is empty and we are not in inversion mode (-v). **Example of Incorrect Fix**: Skip printing if match is empty even if in inversion mode (*Regression* because it breaks inversion mode).

Fig. 1. Complete list of errors and their average debugging time, difficulty, and patch correctness, with human-generated explanations of the runtime actions leading to the error, and examples of correct and incorrect fixes, sorted according to average debugging time (zoom required).