R2Fix: Automatically Generating Bug Fixes from Bug Reports

by

Chen Liu

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2012

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Many bugs, even those that are known and documented in bug reports, remain in mature software for a long time due to the lack of the development resources to fix them. We propose a general approach, R2Fix, to automatically generate bug-fixing patches from free-form bug reports. R2Fix combines past fix patterns, machine learning techniques, and semantic patch generation techniques to fix bugs automatically. We evaluate R2Fix on three large and popular software projects, i.e., the Linux kernel, Mozilla, and Apache, for three important types of bugs: buffer overflows, null pointer bugs, and memory leaks. R2Fix generates 60 patches correctly, 5 of which are new patches for bugs that have not been fixed by developers yet. We reported all 5 new patches to the developers; 4 have already been accepted and committed to the code repositories. The 60 correct patches generated by R2Fix could have shortened and saved an average of 68 days of bug diagnosis and patch generation time.

Acknowledgements

I would like to take the opportunity to express my deepest gratitude to my supervisor Prof. Lin Tan. During the study, she supported me in every aspect. I would like to thank for her enthusiasm and unwavering support, for the numerous useful guidance, discussions, feedback and encouragement. The things I learned from her will be extremely beneficial for my future development.

I am thankful to readers of the thesis, Prof. Patrick Lam and Prof. Mahesh V. Tripunitara, for spending their valuable time to review the thesis and give valuable comments.

Thanks to our research group members, especially Tian and Jinqiu. We have been in the group together for more than one year. I enjoyed discussing with them about topics in scientific research. Thank them for inspirations and good ideas.

Lastly, and most importantly, I would like to acknowledge my family. My dear mother, the first teacher and the role model in my life, gives me confidence to explore new things, especially in a different country far away from my homeland. Thanks to her endless support, sacrifice and patience. My dear father comes next. He taught me how to develop interests in a scientific area. He taught me how to overcome difficulties in study and how to solve problems in daily life. To them I dedicate the thesis.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Everyday, an overwhelming number of bugs are reported. For example, the Mozilla bug database [4], with a total of 670,359 bug reports, receives an average of 135 new bug reports daily. The corresponding bugs hurt software reliability and security, which are not improved until the bugs are fixed.

The process of generating a fix is difficult and time-consuming. Upon receiving a bug report, developers *diagnose* the root cause of the bug, *produce a patch* that can fix the bug, and *commit* the patch to the source code repository. We combine the first two steps (diagnosis and patch generation) under the label of *fixing* a bug, which is the focus of this paper. Developers' bug-fixing process is primarily manual; therefore the time required for producing a fix and its accuracy depend on the skill and experience of individuals.

Figure 1.1(a) shows a Linux kernel buffer overflow/overrun bug report. The developers first need to understand this bug report by reading the relevant code together with this report: the buffer `state` contains only 4 bytes, but 5 bytes, "`off␣\0`", was written to the buffer, where ␣ denotes one space character and the single character '`\0`' is needed to mark the end of the string. The developers then need to identify the root cause: why are more than 4 bytes are assigned to the buffer? Should 5 bytes be allocated instead; should developers assign only 4 bytes to the buffer `state`; did the developers forget to check if the array is long enough to hold the content before the assignment; or was the bug by more complex reasons? Developers often spend days, weeks, or even months diagnosing the root cause of a bug by reading the relevant source code, using a debugger to observe and modify the program execution on different inputs, etc. After a developer determines the root cause, typically the developer needs to figure out how to modify the buggy code to fix the bug, check out the buggy version of the software, apply the fix, and generate the patch.

The result of this challenging and time-consuming process by developers for bug 11975 is the patch in Figure 1.1(b). The patch deletes the line that writes 5 bytes to buffer `state` (denoted by `- strcpy(state, "off␣");`), and adds a new line to write only 4 bytes to `state` (`+ strcpy(state, "off");`), which fixes the overflow bug.

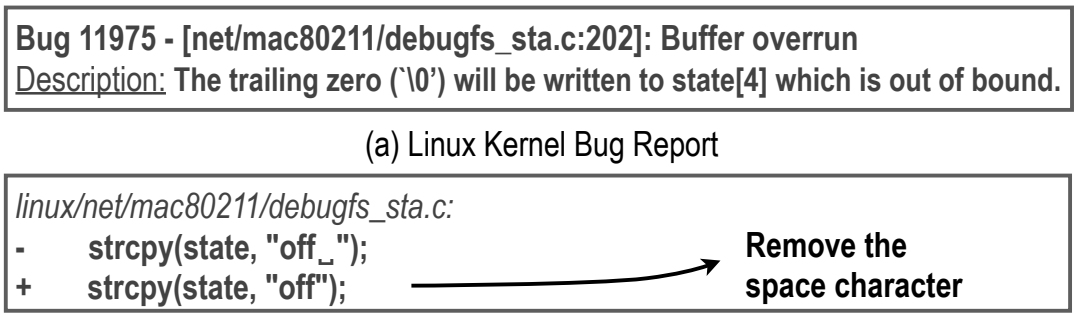


Figure 1.1: Converting a bug report to a patch. "-" denotes a line to be deleted; "+" denotes a line to be added; and "␣" is one space character.

Developers often need to fix more bugs than their time and resources allow [6]. Although developers spend almost half of their time fixing bugs [27], average bugs take *years* to be fixed [12, 25].

Therefore, support to make it easier and faster for developers to fix bugs is in high demand. The capability to automatically generate patches (e.g., Figure 1.1(b)) from bug reports (e.g., Figure 1.1(a)) will: (1) save programmers' time and effort in diagnosing bugs and generating patches, allowing developers to fix more bugs or focus on other development tasks (Section 5.3); and (2) shorten the bug-fixing time, thus improve software reliability and security. (Section 1.3).

1.1 Ideal Goal vs. Realistic Goal

Ideally, we want to automatically generate patches for all bug reports. Realistically, it is impossible. In fact, only 16.7–33.5% of bug reports in the Linux kernel, Mozilla, and Apache bug databases are fixed (Table 4.1). This is because, many bug reports are invalid, unreproducible, incomplete, etc. Even among bugs that can be fixed, some are too complex to be fixed automatically because they require redesign of the algorithm, addition of new features, etc.

Therefore, a realistic goal is to automatically generate patches for relatively simple bugs, e.g., Bug 11975 in Figure 1.1. Such bugs are suitable candidates for automation because the bug

reports contain useful information, e.g., the buggy file and the symptom, even the information in the form of natural language.

As it may take less time for developers to fix simpler bugs, the time savings of automatically generating patches may be small (e.g., the bug in Figure 1.1 was fixed in about 20 hours after the submission of the initial bug report). However, it is still beneficial to automatically fix these relatively simple bugs, because (1) many simple bugs have severe impact, e.g., causing security vulnerabilities; and (2) any time and effort saving in fixing bugs should be valuable.

The bugs R2Fix fixes are simple. But the patches generated are not limited to simple patches. Figure 5.1 (a) shows an automatically-generated complex patch (details in Section 5.1).

1.2 Challenges

Automatically generating patches from bug reports is extremely challenging. First, many bug reports do not contain enough information for a developer to understand how to fix the bug. A developer needs to read the relevant code to find the exact cause and then generate a patch to fix the bug. In order to automatically generate patches from a bug report, we must *automatically recover the missing diagnostic information* without consulting the developers or the reporter.

Second, in order to fix a bug, we need to know its root cause. For example, we need to know that the bug report in Figure 1.1(a) describes a buffer overflow bug. A simple grep of "buffer overflow" in the bug report produces poor results (Section 3.2). We need more accurate techniques to obtain the root cause from a bug report.

1.3 Contributions

As a first step to address this challenging task, we propose *R2Fix*, a novel and general technique to automatically generate bug-fixing patches from relatively simple free-form bug reports. As different types of bugs require different types of fixes, R2Fix generates bug fixes (also referred to as patches) by bug type (e.g., buffer overflows, or memory leaks). R2Fix analyzes bug reports, determines the bug types, and generates patches for developers to verify. When R2Fix cannot generate a patch for a bug report, developers can follow the normal procedure of addressing the bug report, in which case R2Fix adds no extra work to the developers.

We use novel techniques to address the challenges described in Section 1.2. First, to recover missing information, we *leverage past bug fix patterns to automatically diagnose bugs and generate fixes*. For example, if most buffer overflows are fixed by two common fix patterns, and

we generate two patches according to the two patterns for an overflow bug, then one of the two patches is likely to be a correct patch (Details in Section 2 and Section 3). In addition, R2Fix's diagnostic capability can often narrow the candidate set down to one patch (Section 3). Our fix pattern study (Section 2) confirms that many bugs are fixed by a few common fix patterns. Second, we apply machine learning techniques to automatically identify the root causes from bug reports.

We evaluate R2Fix on three large and popular software projects—the Linux kernel, Mozilla, and Apache—for three important types of bugs: buffer overflows, null pointer bugs, and memory leaks. For bug reports that R2Fix can generate patches for, we compare the patches with developer-generated patches if they can be identifies. Otherwise, we report R2Fix-generated patches to developers and let developers verify the patches.

In total, for 819 randomly sampled fixed and unfixed bug reports, R2Fix automatically generates patches for 80 verifiable bug reports, 60 of which are patched correctly by R2Fix, a precision of 75.0%. Among the 60 patches, 5 are new patches generated for unfixed bug reports. After we reported the 5 new patches to the developers, **4 are accepted and committed to the source code repositories** immediately, while the other await developer confirmation.

If developers had applied R2Fix as the bug reports were submitted, **R2Fix could have shortened the bug-fixing time from an average of 68 days to 0**. The *bug-fixing time*, consisting of the diagnosis time and patch generation time, is defined as the time from when a bug report is opened to when a correct patch to fix the described bug is submitted. As the fixing time of 68 days does not include the time for developers to commit the patch to the code repository, the potential speedup of 68 days is a good approximation of how much time R2Fix saved on diagnosing and generating the fixes.

Among the 60 automatically generated correct patches, 21 fix potential security vulnerabilities (buffer overflow bugs), and **3 patches fix confirmed security vulnerabilities** in the Linux kernel and Apache recorded in the National Vulnerability Database (NVD) [5]. If R2Fix were applied to these security bugs, the vulnerability window could have been shortened by a minimum of 8 days to up to 55 days.

Although R2Fix automatically generates correct patches for many bug reports, they constitute a small percentage (<1%) of all fixed and unfixed bug reports in the evaluated projects. We briefly discuss two reasons here, while Section 5.1 presents a detailed discussion and Section 6 suggests approaches to increase this percentage. First, we randomly sampled 819 bug reports for evaluation. Second, we only evaluate R2Fix on three types of bugs as a proof of concept. Our coarse estimate suggests that we could extend R2Fix to handle 17.2% of all fixed bug reports in the three evaluated projects. Note that even 1% of fixed bug reports in the three evaluated projects is amount to 19,600 bug reports; the time savings on fixing them would be highly considerable.

This thesis makes the following contributions:

- We propose to automatically generate patches from free-form bug reports. As a proof of concept, our prototype, R2Fix, automatically generates correct patches that are *directly* applicable to the faulty software to fix bugs.
- Our prototype R2Fix currently handles three types of bugs, which has significant impact: (1) R2Fix could have **shortened the bug-fixing time** from an average of 68 days to 0; (2) R2Fix fixed 21 potential security vulnerabilities, and 3 patches **fix confirmed security vulnerabilities**, shortening vulnerability window; and (3) some developers from the three evaluated projects consider that R2Fix can **save developers' time** in fixing bugs (Section 5.3). In addition, it is promising to extend R2Fix to handle 17.2% of all fixed bug reports in the three evaluated projects based on our coarse estimate.
- We perform a detailed bug fix pattern study (Section 2) and leverage the results with R2Fix to automatically recover missing diagnostic information in the bug reports for patch generation. Our study shows that a significant proportion (27.3–86.6%) of buffer overflows, null pointer bugs, and memory leaks are fixed by a few simple fix patterns.

Chapter 2

A Study of Fix Patterns

There are two main reasons to study bug fix patterns. First, bug fix patterns affect the feasibility and applicability of automatic patch generation from bug reports. If many bug fixes share a small set of fix patterns, then it is feasible to automatically generate patches based on the common fix patterns. Second, fix patterns can recover diagnostic information missing from a bug report (Chapter 3).

We study three *important and dominant* types of bugs—buffer overflows, null pointer bugs, and memory leaks. Buffer overflows remain the single largest contributing factor to reported security vulnerabilities [16]. Null pointer bugs and memory leaks are the top two major types of memory bugs in modern software [30]. These bugs can cause crashes, corrupt persistent data, degrade performance, and open doors for severe security attacks.

2.1 Data Collection

To conduct a comprehensive study of fix patterns, we collect a reasonably large sample of each type of bugs, and manually identify the common fix patterns. To ensure the identification of correct fixes, we only study *closed fixed* bug reports whose fixes can be identified from the reports or the version control systems. In total, we collect 51 buffer overflow bug reports, 91 null pointer related bug reports, and 41 memory leak bug reports from the Linux kernel and Mozilla bug databases using the following two approaches.

We first randomly sample 636 closed fixed bug reports from the two bug databases. We manually read them and find 6 buffer overflows, 10 null pointer related bugs and 11 memory leaks, which are too few for a comprehensive study of fix patterns.

Pattern	L(%)	M(%)
LongerBuf	60.0	19.4
FewerByte	13.3	19.4
MdBound	13.3	9.7
Total	86.6	48.5

(a) Overflow

Pattern	L(%)	M(%)
AddCheck	47.6	53.3
MvCheck	4.8	10.0
RmCheck	9.5	6.7
Total	61.9	70.0

(b) Null Pointer

Pattern	L(%)	M(%)
AddFree	53.8	22.8
MvFree	0	4.5
Total	53.8	27.3

(c) Memory Leak

Table 2.1: Common Fix Patterns. L denotes Linux and M denotes Mozilla.

Therefore, we use keywords related to each bug type to collect additional closed fixed bug reports for each bug type. The keywords used for finding potential buffer overflows are "buffer", "overflow", "overwrit", "overrun", "overlap", while the keyword for null pointer related bugs is "null pointer". For memory leak bugs, keywords "memory leak" and "leak" are used. Bug reports that contain these keywords are not necessarily a bug of these bug types, so we manually read a random sample of 100–240 for each bug type (in proportion to the total number of bug reports of each bug type) to discover additional 45 buffer overflow bugs, 81 null pointer bugs, and 30 memory leak bugs.

2.2 Fix Pattern Study Results

Interestingly, Table 2.1 shows that *many bugs are fixed by a few simple fix patterns and the fix patterns for the same type of bugs are the same in different software*. Specifically, Table 2.1(a) shows that 48.5–86.6% of the buffer overflow bug reports in the Linux kernel and Mozilla are fixed by three common bug fix patterns—allocating a longer buffer (LongerBuf), assigning fewer bytes to a buffer (FewerByte), and modifying the bounds check conditions (MdBound). Table 2.1(b) demonstrates that 61.9–70.0% of the null pointer related bug reports are fixed by adding a null check before dereferencing the pointer (AddCheck), moving the null check before dereferencing the pointer (MvCheck), and removing unnecessary null checks (RmCheck). Note that null pointer bugs are not limited to null pointer dereferences. We also count unnecessary null check as null pointer bugs in this thesis. Therefore, RmCheck belongs to the fix patterns for null pointer bugs. Table 2.1(c) shows that 27.3–53.8% of the memory leaks are fixed by adding code to free memory (AddFree) and moving the memory releasing code so that it frees the memory in all paths (MvFree).

The results demonstrate the viability of (1) leveraging these fix patterns to automatically generate patches from bug reports; and (2) reusing the fix patterns to automatically generate patches for other software. In fact, we use the fix patterns learned from the Linux kernel and Mozilla to

Pattern	Subpattern	Param
Overflow(6): FewerByte	- strcpy(BUF,EXPR); + strlcpy(BUF,EXPR, sizeof(BUF));	none
	- sprintf(BUF,FMT,EXPR); + snprintf(BUF, sizeof(BUF),FMT,EXPR;	none
	+ int hlen; ... - sprintf(BUF,FMT,EXPR) + hlen = snprintf(BUF, sizeof(BUF), + FMT,EXPR); [[... - sprintf(BUF,``%s FMT2'',BUF,EXPR2); + hlen += snprintf(BUF+hlen, + sizeof(BUF)-hlen,`` FMT2'',EXPR2);]]+	none
Nullptr(11): AddCheck	+ if (PTR) FNC(...,PTR,...);	FNC & PTR

	FNC(...){
	+ if (PTR)
	PTR->FLD = EXPR;
	}
		FNC & PTR

RmCheck	TYPE I = PTR->FLD; ... - if (!PTR) {...}	none
Leak(2): AddFree	TYPE *PTR = ...; ... if(EXPR) + { delete PTR; return ...; + }	none

Table 2.2: Fix Pattern Examples. The numbers are the total number of fix patterns for each bug type.

effectively generate patches for Apache; and the same fix patterns are used for all three projects (except that `PR_snprintf` and `PL_strncpy` are used for Mozilla, which are Mozilla's own ver-

sions of the standard `snprintf` and `strlcpy` calls used for the Linux kernel and Apache).

Table 2.2 shows examples of fine-grained bug fix patterns (also referred to as fix subpatterns). The complete list of fix subpatterns is shown in Appendix (see the end of this paper). For example, six (6) subpatterns are used for buffer overflows, two of which are shown in Table 2.2.

To use these fix patterns for patch generation, we need to know the *pattern parameters*, such as the pointer name, so that we know for which pointer we need to add a null check. Therefore, each fix pattern is shown with the pattern parameters required (column "Param"). Other uppercase names represent what is extracted from the code during the fix pattern matching process. For example, the first FewerByte subpattern searches for `strcpy(BUF, EXPR)` in the target code file (details in Section 3.4), where 'BUF' matches a program identifier and 'EXPR' matches an expression, and replaces it with `strlcpy(BUF, EXPR, sizeof(BUF))`. The first AddCheck subpattern means adding a check—`if (PTR)`, if identifier `PTR` is used as a function parameter. To map these fix patterns to all equivalent code by semantic matching, 'TYPE' matches a valid type; 'EXPR' and 'EXPR2' match valid source code expressions; 'BUF', 'I', 'PTR', 'FLD', and 'FNC' match identifiers; and 'FMT' and 'FMT2' are format strings. '[[]]+' indicates the content should repeat one or more times.

Chapter 3

R2Fix Overview

Upon receiving a bug report, R2Fix analyzes the bug report, determines the bug type, and generates possible patches to fix the bug. As bug reports describe bugs in different software versions, R2Fix automatically identifies the buggy version and generates patches for that version.

R2Fix uses common bug fix patterns to automatically generate patches with high accuracies of 74.1–77.3%, even if some diagnostic information is missing in the bug reports. In addition, the above process enables R2Fix to diagnose a bug report by narrowing down to one or two relevant patches. For example, the second FewerByte subpattern does not generate a patch for the bug report in Figure 1.1(a) because R2Fix automatically detects that the relevant source code does not contain a call to `sprintf`. R2Fix generates on average 1.33 patches per bug report, which is much lower than the number of fix patterns used (6, 11 and 2 for the three bug types). The results show that R2Fix is precise in generating patches, so that developers do not need to wade through many R2Fix-generated patches to find the correct patch.

3.1 R2Fix Architecture

Figure 3.1 shows that R2Fix has three analysis steps: (1) *Bug Classifiers*, or *Classifiers* for short, parse and classify bug reports according to the target bug types, and retains only bug reports that are classified as the target bug types, referred to as *Candidate Bug Reports*, for the next two steps; (2) A *Pattern Parameter Extractor*, or *Extractor* for short, analyzes the candidate bug reports and source code to extract pattern parameters—pointer names, buffer lengths, etc.; and (3) A *Patch Generator*, or *Generator* for short, uses the pattern parameters, the fix patterns for each target bug type, and the source code repository to automatically generate patches. This section provides

a brief summary of the three components, while the details are described in Section 3.2, 3.3, and 3.4 respectively. Chapter 6 illustrates how to use test cases to validate patch correctness.

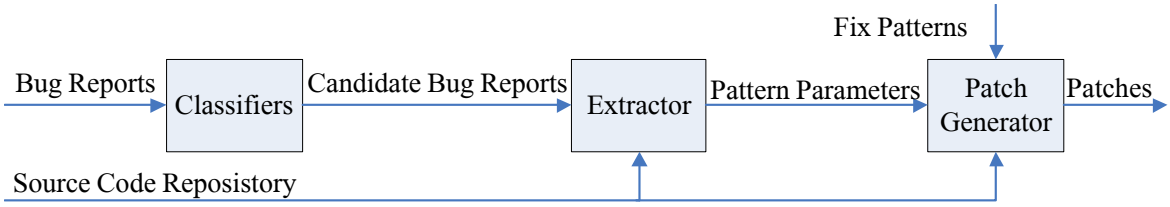


Figure 3.1: The Architecture of R2Fix

3.1.1 Bug Classifiers

Following machine learning methods [33], classifiers are built using a small *training set* of manually labelled bug reports. We then apply the classifiers on all bug reports to identify candidate bug reports. The majority (94.2%) of the candidate bug reports come from the *unlabelled* bug reports that are not used in our training. One independent classifier is built for each bug type; therefore, building classifiers for new bug types do not affect the accuracy of the existing classifiers.

The effort of manually labelling bug reports is only required ***once per bug type***, because our results show that classifiers trained on bug reports of some representative software (Mozilla and the Linux kernel) can classify bug reports of other software (Apache) with high accuracies (96.6–98.8%) and precisions (0.86–0.90). In contrast, the effort of manually fixing a bug report is required for each bug report. Therefore, R2Fix can save developers' time in fixing bugs in the long run.

3.1.2 Pattern Parameter Extractor

Although the pattern parameters are different for different types of bugs, some parameters are common across different bug types, such as pointer names and function names. R2Fix employs a *general technique that uses the source code and bug reports together to extract pattern parameters*.

3.1.3 Patch Generator

For a candidate bug report, the Patch Generator applies all applicable fix subpatterns for the bug type independently of the target version of the *target file*. Fix patterns with parameters are

applicable to a bug report, only if the Extractor can extract the required parameters from that bug report. In addition, a patch is generated only when an applicable fix pattern can find a match in the target file. These steps ensure that only relevant patches are generated for a bug report for high patch generation accuracy. Patches generated from multiple fix patterns for the same bug report are independent. Developers will select the most appropriate patch to fix the bug, which should be easy because (1) R2Fix generates only 1.33 patches per bug report on average (Table 5.1); and (2) developers quickly selected the correct patch after we reported them (Section 5.1).

3.2 Bug Classifiers

3.2.1 Keyword Search versus Classification

A straightforward approach to identify bug reports of a given bug type is to search bug reports for keywords such as "buffer overflow", and "null pointer". This approach has a low precision. For example, only 19% of the bug reports with keywords such as "buffer overflow" describe buffer overflow bugs. In addition, this approach is not general, as different keywords may be needed for searching for the same type of bug reports in different software. R2Fix classifiers can identify buffer overflow, null pointer, and memory leak bug reports with much higher precisions (0.82–0.96 in Table 5.2), thus generates much fewer false positives in the candidate bug reports.

3.2.2 Bug Report Parsing and Classification

A typical bug report contains: a *Summary*, the *Report Submission Time*, the *Version* of the buggy software, the *Component*, the *Priority* and *Severity*, a free-form *Initial Report* of the bug, and zero to many free-form *Follow-up Reports* following the Initial Report. Users and developers can use follow-up reports to discuss the bug, asking for additional information, etc.

R2Fix classifiers use all of the fields above *except the follow-up reports, because we want to apply R2Fix as soon as a bug is reported to maximize the time that R2Fix can save for developers in fixing bugs*. If developers spend time diagnosing and patching a bug, they may post the diagnostic information in a follow-up report. While all results presented in this paper are produced without using follow-up reports, developers can always use R2Fix to analyze the entire bug reports including all the follow-up reports to generate more patches and more precise patches. Any bug report whose initial report contains a patch attachment is filtered out in this parsing step, as it is unnecessary to apply R2Fix if a patch has already been generated by developers.

We use the same bug reports from Section 2.1 to build the training sets. We use the bag-of-words model to represent the summary and the follow-up reports. Two-level classification usually yields a better accuracy because the second-level classifier is applied on the filtered subset [26]. Thus we deploy a two-level classification approach, which first determines whether the bug reports describe real bugs. A second level classifier, built from the same bug reports but with labels indicating the specific bug types, classifies the bug reports into target bug types.

3.3 Pattern Parameter Extractor

3.3.1 What pattern parameters to extract?

Two types of pattern parameters are essential for patch generation: generic parameters and bug-type-specific parameters. Generic parameters, required for all bug types, include the file name and the version number of the faulty software. Different types of bugs need different bug-type-specific parameters, which are shown in the column "Param" of Table 2.2. For example, the AddCheck fix pattern needs two bug-type-specific parameters—"FNC" and "PTR". "PTR" is the null pointer passed as a parameter to the function "FNC".

3.3.2 How to extract the pattern parameters?

It is trivial to extract the buggy version number from the version field. However, the version field is often empty. For these bug reports, R2Fix automatically extracts the report submission time, and use it to check out the faulty version from the code repository. This simple analysis has an accuracy of 96.4%. R2Fix uses regular expression matching to find source file names that end with .c, .cpp, .h, etc. in a bug report.

Most of the bug-type-specific parameters are program identifiers, e.g., function names, buffer names, etc. To extract such identifiers, for each word in a bug report, the Extractor searches the word in the source code to see whether it is a function name, a buffer name, etc. This approach is *general* and independent of bug types or the format of the bug reports. In addition, if a buffer declaration, e.g., `int a[10]`, is found in the bug report using our regular expression matching, we consider it as a buffer name. If multiple function names and file names are extracted, our Extractor keeps only the first file name and the first two function names. These heuristics are not perfect, but they work well in practice.

3.4 Patch Generator

To ensure patch generation accuracy, the fix patterns are only applied to the *target file*, not to the entire code base. In particular, if generic parameters, i.e., file name and version information, cannot be extracted by the Extractor, no patch is generated for the bug report.

We use examples to explain how the Patch Generator automatically generates patches. Given the bug report in Figure 1.1(a), which has been classified as an overflow bug, the Generator applies all applicable overflow fix subpatterns to the target version of the target file, *linux/net/mac80211/debugfs_sta.c*. For example, both overflow subpatterns in Table 2.2 are applicable for this bug report since no pattern parameters are required. However, only the first subpattern finds a match in *debugfs_sta.c*.

In the target file, the Generator searches for `strcpy(BUF, EXPR);`, where `strcpy` matches a function whose name is exactly `strcpy`, `BUF` matches any program identifier, and `EXPR` matches any code expression. When a match `strcpy(state, "off␣");` is found, the Generator will generate the following patch, which is semantically equivalent to the patch in Figure 1.1(b):

```
- strcpy(state, "off␣");
+ strlcpy(state, "off␣", sizeof(state));
```

Some fix subpatterns, e.g., the AddCheck subpattern in Table 2.2, require pattern parameters. If two bug-type-specific parameters, the pointer name `reply_msg` and the function name `c2_errno`, are extracted for a bug report, the concrete fix pattern for the AddCheck subpattern is:

```
+ if (reply_msg)
        c2_errno(..., reply_msg, ...);
```

where `...` matches any code segment.

Although adding a null check before a pointer dereference is a known way to fix null pointer bugs, it is impractical to add a null check before all pointer dereferences due to the high runtime overhead. Our fix patterns are applied only to a reported bug; therefore, R2Fix is unlikely to add unnecessary null checks.

Due to code isomorphisms and the differences in spacing, regular expression based matching is insufficient to identify all equivalent code segments. We leverage a successfully used tool Coccinelle [35] to perform semantic match of the fix patterns. For example, the RmCheck subpattern in Table 2.2 needs to search for `if (!PTR)`, but a regular expression search misses the semantically equivalent code `if (PTR==NULL)`, while our Coccinelle-based matching can identify it as a match.

Chapter 4

Experimental Methods

4.1 Evaluated Software

We apply R2Fix on all bug reports in bug databases [1, 3, 4] of three large open source projects (Table 4.1) to automatically fix three important types of bugs. Mozilla is a mix of C and C++, while the Linux kernel and Apache are purely in C.

Software	**BR**	**Fixed BR**	**Open BR**	**LOC**	**Description**
Linux	16.4K	5.5K	2.6K	11.9M	Operating System
Mozilla	599.8K	189.1K	67.1K	5.0M	Browser Suite
Apache	5.4K	0.9K	1.0K	0.3M	Web Server

Table 4.1: Evaluated software. BR is the total number of bug reports, and LOC is lines of code.

4.2 Closed Fixed Bug Reports and Open Bug Reports

We apply R2Fix on *all* bug reports in the evaluated bug databases. We first apply R2Fix on all closed fixed bug reports to evaluate its accuracy by comparing the generated patches against the developer-generated patches. In addition, we run R2Fix on all open bug reports so that we can submit the patches to the developers to save their time and effort. Many open bug reports contain correct developer-generated patches, but simply are not marked as closed yet. Therefore, we separate *open unfixed bug reports* from these *open fixed bug reports*: for open unfixed bug reports, we report R2Fix-generated patches to the developers for their verification.

4.3 Evaluation Measures

We define the patch precision PPrecision as:

$$PPrecision = \frac{\text{Total Number of Correctly Patched Bug Reports}}{\text{Total Number of Verifiable Patched Bug Reports}}$$

Patched bug reports are bug reports that R2Fix can generate at least a patch for. We consider a bug report *correctly patched* if at least one of the R2Fix-generated patches is the same or semantically equivalent to the developer-generated patch. Therefore, we can only evaluate the PPrecision on patched bug reports whose developer-generated patches can be found, referred to as *verifiable patched bug reports*. For an open unfixed bug report, we consider it verifiable if two authors of this paper independently consider the R2Fix-generated patch correct. We still report the patches to developers and wait for their confirmation.

For classification, we measure the standard accuracy, precision, recall and F1. Accuracy is defined as:

$$Accuracy = \frac{\text{Total Number of Correctly Classified Bug Reports}}{\text{Total Number of Bug Reports Given for Classification}}$$

Precision is defined as ($P = \frac{T_+}{T_+ + F_+}$), recall is ($R = \frac{T_+}{T_+ + F_-}$), and F1 is ($F1 = \frac{2PR}{P+R}$), where T_+, T_-, F_+ and F_- are, true positives, true negatives, false positives and false negatives respectively. Specifically, instances in the test set will be labeled by either "Yes" or "No" after the classification. True positives are instances that are correctly classified as "Yes". Similarly, true negatives are instances that are correctly classified as "No". False positives and false negatives refer to those incorrectly labeled instances.

4.4 Two Classification Experiments

The first experiment combines bug reports of the same type from all evaluated software to form one training set to build precise classifiers. The standard 10-fold cross-validation is used. We experiment with three algorithms in Weka [44], i.e., Decision Tree, Support Vector Machine (SVM), and Bayesian Logistic Regression (BLR). We pick classifiers with the best precision from the cross-validation results. We then apply these classifiers to the entire bug databases.

Specifically, BLR is used for classifying buffer overflows with threshold 0.4 and tolerance 5.0E-4; SVM with a linear kernel is used for null pointer bugs and memory leaks. The same algorithms and parameters are used for all three projects.

To demonstrate that a model trained from representative software can be used to classify bug reports in another software project, we conduct *cross-software classification* experiments. We use the model built from the Linux kernel and Mozilla bug reports to classify Apache bug reports.

4.5 Developer Feedback

To obtain feedback about the usability and benefits of patches automatically generated by R2Fix, we sought answers to two questions:

Q-1 Would a patch automatically generated by R2Fix save developers' time in fixing the bug?

Q-2 Would a patch automatically generated by R2Fix
prompt a quicker response to the bug report?

We send to developers emails that contain a link to a bug report, the corresponding R2Fix-generated patch, and the two questions above (including possible answers and space for the participants to write their own answers). The bug reports are randomly sampled from what R2Fix can correctly generate a patch for.

Chapter 5

Results

5.1 Patch Generation Results

		Closed Fixed Bug Reports						Open Bug Reports					
Type	Software	Candidate /Sample	Patched Reports	Verifiable Patched	Correct	PPrecision	Patches-PerBug	Candidate /Sample	Patched Report (Fixed/ Un-fixed)	Verifiable Patched (Fixed/ Unfixed)	Correct (Fixed/ Unfixed)	PPrecision	Patches-PerBug
Overflow	Linux	33/33	13	10	8	80.0%	1.15	13/13	6(5/1)	5(4/1)	4(3/1)	80.0%	1.16
	Mozilla	89/89	6	5	5	100.0%	1.00	27/27	4(2/2)	3(2/1)	2(1/1)	66.7%	1.00
	Apache	9/9	3	3	1	33.3%	1.33	5/5	1(1/0)	1(1/0)	1(1/0)	100.0%	1.00
NullPtr	Linux	56/56	11	11	7	63.6%	1.82	48/48	8(4/4)	7(3/4)	6(3/3)	85.7%	2.00
	Mozilla	969/100	6	6	4	66.7%	1.50	188/100	4(4/0)	3(3/0)	2(2/0)	66.7%	2.00
	Apache	21/21	2	2	1	50.0%	1.50	2/2	0	0	0	0	0.00
Leak	Linux	40/40	20	20	16	80.0%	1.00	35/35	3(3/0)	3(3/0)	2(2/0)	66.7%	1.00
	Mozilla	1,568/100	1	1	1	100.0%	1.00	483/100	0	0	0	0	0.00
	Apache	18/18	0	0	0	0	0.00	23/23	0	0	0	0	0.00
Total		2,803/466	62	58	43	74.1% AVG	1.26 AVG	824/353	26(19/7)	22(16/6)	17 (12/5)	77.3% AVG	1.50 AVG
Closed + Open		**3,627/819**	**88**	**80**	**60**	**75.0%** AVG	**1.33** AVG						

Table 5.1: Overall Results. AVG denotes that the number in the cell is the average.

Table 5.1 shows R2Fix's automatic patch generation results. "Candidate" is the number of bug reports that are classified as one of the three bug types. As thousands of Mozilla bugs are classified as candidate Nullptr and Leak bug reports, we *randomly* sampled 100 from each to verify the accuracy of R2Fix in patch generation. The sample sizes are shown in column "/Sample". "Patched Report" is the number of bug reports that R2Fix can generate at least a patch for. "Verifiable Patched" is the number of verifiable patched bug reports (defined in Section 4). "Correct"

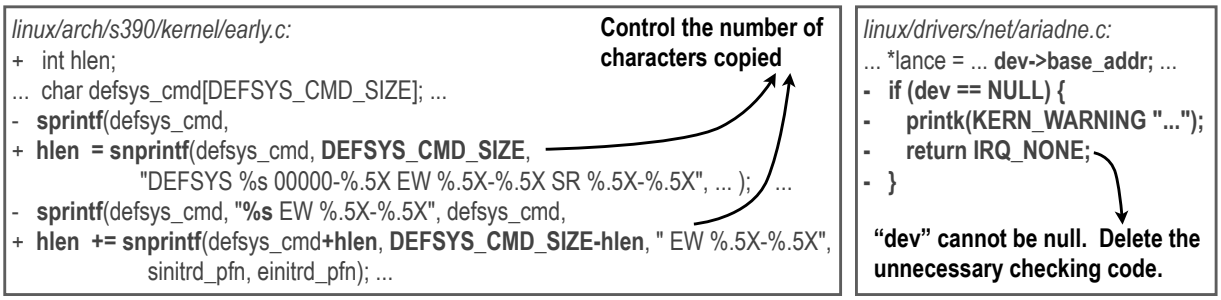
is the number of bug reports for which R2Fix generates at least a correct patch. "PatchesPerBug" is the average number of patches that R2Fix generates for a bug report. "(Fixed/Unfixed)" shows the breakdown into open fixed bug reports and open unfixed bug reports. Row "Closed + Open" sums up the numbers for closed fixed and open bug reports.

R2Fix automatically generates *correct* patches for 60 bug reports with a precision of 75.0%, 43 of which are generated for closed fixed bug reports, and 17 of which are for open bug reports. On average, R2Fix generates 1.33 patches per bug report. The results show that R2Fix is effective, accurate, and precise in automatically generating patches.

R2Fix successfully generated 5 new patches for open unfixed bug reports, 4 of which have been accepted and committed to the code repositories soon after we reported. R2Fix shortened the bug-fixing time for these bugs, as these bugs would not have been fixed as fast had we not reported the R2Fix-generated patches. This is because developers respond quicker to a bug report with a patch attached [41], which is confirmed by developers' feedback (Section 5.3), and our experience that *several bugs that were open for over a year were fixed right away after we reported our automatically-generated patches*. In addition, if developers had applied R2Fix to bug reports as they are being reported, R2Fix could have shortened the average fixing time by 68 days and saved developers time and effort in diagnosing and fixing these bugs.

Among the 60 correct patches, 21 fix potential security vulnerabilities (buffer overflow bugs), including **3 patches fixing confirmed security vulnerabilities** in NVD [5]. If R2Fix were applied to these security bugs, the vulnerability window could have been shortened by 8–55 days.

Although R2Fix generates correct patches for many bug reports, they constitute a small percentage (<1%) of all closed fixed and open bug reports in the evaluated projects for the following reasons. First, we only evaluate R2Fix on three types of bugs as a proof of concept (see Section 1.3). Second, we randomly sampled 819 out of the 3,627 candidate reports (22.6%), because it is time-consuming to find developer-generated patches to verify the R2Fix-generated patches. Note that this manual patch process of identifying developer-generated patches is only for evaluation purposes. We could generate and verify more patches for the rest of the bug reports given more time. Third, R2Fix can only generate patches for a portion of the bugs of the three bug types (10.7% = 88/819 in Table 5.1), as other bug reports do not contain enough information, require fixes not covered by our fix patterns, etc. Lastly, for some bug reports R2Fix can generate a patch, the bug reports and the code repositories do not contain enough information for us to find the developer-generated patches, so that we cannot evaluate the correctness of R2Fix; hence, these are excluded from our evaluation. Note that this is not a limitation of R2Fix; if the developer-generated patches are available, we could have evaluated our R2Fix-generated patches against them. All these reasons combined contribute to the small percentage. Nonetheless, our technique generate 60 correct patches with a precision of 75.0%.



(a) Patch for Overflow Bug 12965 (b) Patch for Nullptr Bug 13853

Figure 5.1: R2Fix automatically generated the two patches, both of which have been accepted and committed to the code repository by the kernel developers soon after we reported them. The generated patches are directly applicable to the faulty code, but are simplified for illustration.

5.1.1 More Patch Examples

Figure 5.1 (a) shows a relatively complex R2Fix-generated patch. After we reported the patch, the Linux kernel developers accepted the patch and committed it to the kernel git repository. Although the bug report missed describing one location of the bug, i.e., one misused `sprintf`, R2Fix was able to automatically find the missing misuse of `sprintf` and fix it, which is an advantage of R2Fix over manually generating patches.

This bug was detected by the Cppcheck tool [2]. However, R2Fix's fix patterns and semantic-based patch generation process are still needed for fixing tool-detected bugs. In addition, as most reported bugs in bug databases are not detected by a tool and 27 out of the 60 bugs that R2Fix fixes correctly are not detected by a tool, it is mandatory for R2Fix to analyze bug reports written in natural language to automatically generate patches.

Mozilla developers have also accepted and committed an R2Fix-generated patch for a buffer overflow bug. In addition, R2Fix automatically generated a Linux kernel patch to remove an unnecessary null check because the pointer `dev` cannot be null (Figure 5.1(b)). For the same bug report, R2Fix generates an additional patch that moves the null check before the dereference of `dev`. After we reported both patches to the Linux kernel mailing list, **the developers were able to quickly identify the patch shown in Figure 5.1(b) as correct**. The bug report contains code to point out how to fix the bug, which suggests this is likely to be an easy-to-fix bug; however, the bug has not been fixed for more than a year since its submission. After we sent the patches, the Linux developers fixed it within a day, which suggests that the bug is important enough that the developers fixed it, and that R2Fix *shortened its bug-fixing time*.

Type	Acc(%)	P	R	F1	Size	Pos
Overflow	95.7	0.90	0.48	0.62	794	59
NullPtr	92.6	0.82	0.53	0.64	801	100
Leak	99.2	0.96	0.88	0.92	994	52

Table 5.2: Classification Results. Acc is Accuracy; Size is the training set size; and Pos is the number of positive bug reports (bug reports of the corresponding bug type) in the training sets.

5.1.2 Incorrect Patches

Although R2Fix is accurate, there is still space for improvement. R2Fix may incorrectly generate a patch if an irrelevant buffer name is extracted from the bug report. However, these inaccuracies may be masked, e.g., if no code regarding the irrelevant buffer matches with any fix patterns, no patch will be generated. Our coarse estimation shows that the average number of buffer/pointer names in bug reports that contain a file name is 0.66, which indicates that the probability of an irrelevant buffer/pointer name is low. In the future, we can apply semantic analysis on bug reports to reduce such inaccuracies [45].

Note that even incorrect tool-generated patches can shorten the bug-fixing time [14, 41]. In addition, 14.8–24.4% of developer-generated patches are incorrect [45].

5.2 Classification Results

Table 5.2 shows that R2Fix can identify different types of bug reports automatically with high accuracy (92.6–99.2%) and high precision (0.82–0.96). The recalls are lower, meaning that the R2Fix classifiers miss some candidate bug reports. Our design goal is to favour precision over recall so that patches generated are accurate and precise. If R2Fix misses a candidate bug report, and does not generate any patch for it, R2Fix adds no extra work for the developers.

Using a relatively small set of manually identified positive bug reports (column "Pos"), R2Fix can discover over thousands of candidate bug reports (column "Candidate" of Table 5.1). Numbers in column "Pos" are positive bug reports from all evaluated projects, therefore are bigger than the corresponding numbers in Section 2.1, which are for Mozilla and the Linux kernel only. The cross-software classification results have already been discussed in Section 3.

5.3 Developer Feedback Results

More than half (4 out of 7) of the developers answered that the R2Fix-generated patches can save their time (Q-1) in (1) understanding the bug; and (2) fixing the bug (including generating the patch using *diff*, etc.). Almost all (6 out of 7) developers answered that they would respond quicker to a bug report with a R2Fix-generated patch attached (Q-2).

A concern from the negative answers is about R2Fix's accuracy. This concern can partially be attributed to that we forgot to provide the high accuracy of R2Fix to the developers. We are confident that with an accuracy of 75.0%, R2Fix will save developers' time. Section 6 illustrates how to validate patches to ensure patch correctness.

Although the respondents may not represent all developers, the positive responses from these real developers demonstrate that **some developers find R2Fix-generated patches save their time in fixing bugs** and they are likely to respond faster to the corresponding bug reports. These respondents should represent a large number of developers.

Chapter 6

Discussion

6.1 Applicability and Generality of R2Fix

To understand the applicability of R2Fix, we want to know what percentage of bug reports contain detailed information that R2Fix can leverage. Our coarse estimate shows that 17.2% of fixed bug reports in the evaluated projects contain detailed information, i.e., buffer names, line numbers, function names, or file names, but no patch is attached within one hour of initial bug report submission. Considering that the current implementation of R2Fix targets only three bug types, this suggests that R2Fix has significant applicability to be extended for new types of bugs. Adding new bug types does not increase the number of R2Fix-generated patches per bug report, as fix patterns are tied to a bug type. Specifically, our preliminary study shows that it is promising to extend R2Fix to off-by-one and integer overflow bugs. In addition, we can add more fix patterns to fix more bug reports at the cost of increasing the average number of patches per bug report. Semantic analysis of bug reports and bigger training set can also help.

For bug reports that do not contain enough details, we may leverage techniques such as ESD [46] and BugRedux [24] to reproduce the bugs and collect additional information to help R2Fix generate patches. Although R2Fix may not be directly applicable to some bug types due to the complexity of their fix patterns (e.g., concurrency bugs [21, 32]), R2Fix is effective in shortening the fixing time of many important bugs (including security bugs), and saving developers' time.

There are mainly two reasons for R2Fix to target free-form bug reports instead of stack traces: (1) For bugs that produces stack traces, the bug reports usually also contain the stack traces. Most non-crash bugs do not produce stack traces, thus will only be reported to the bugzilla. (2)

Developers also would like to use the natural language to describe their intuitions about potential bugs to bugzilla without actually executing the program. R2fix is able to analyze these bug reports so that it could increase the chances of fixing more bugs.

6.2 Patch Validation

After a patch is generated, developers often need to validate the correctness of the patch before they commit the patch to the source code repository, through testing, code review, etc. We successfully validated the R2Fix-generated patch for bug 523216 in Mozilla by using the test case attached with the bug report to fix the null pointer bug. The patch generated by R2Fix adds a null check for the pointer before it could be dereferenced. The test case fails on the buggy version and passes after the patch was applied. We could not conduct a large scale validation experiment because Mozilla, Apache, and the Linux kernel do not have readily available test cases written by developers. Note that whether developer-written test cases are available varies from project to project, and it is not a limitation of our approach. In fact, it reveals an advantage of R2Fix: R2Fix can generate patches automatically without test cases, while the previous work [14, 42] cannot without test cases. In the future, we could leverage techniques such as KLEE [8] to generate test cases automatically for patch validation.

6.3 Little Manual Effort Required

It is fully automatic to use R2Fix to generate patches for overflows, null pointer bugs, and leaks in other software. This is feasible because our results show that (1) the same type of bugs share similar fix patterns in different software; and (2) classifiers trained from representative software can accurately classify bug reports in other software. Extending R2Fix for other types of bugs requires new training data and the relevant fix patterns. Our preliminary results show that it is promising to automatically extract common fix patterns from developer-generated patches and their corresponding bug reports.

6.4 Threats to Validity

The 68 days of average fixing time does not take into account the time for developers to select the correct patch if multiple patches are generated. In addition, it may contain time when developers

are not actively working on the bug reports. However, the 68 days should be a good approximation of how much time R2Fix can save developers. First, it took us only about 5 minutes on average to select the correct patch, and developers should be able to do it faster. As discussed in Section 5.1, developers were able to quickly pick the correct patch in Figure 5.1(b) from the two patches we sent. Recall that, for 67 bug reports, R2Fix generates only one patch with a precision of 70.1%. Developers can choose to use R2Fix only for these bug reports. Second, for 56% of the bug reports, the developers have been actively discussing the bugs (at least once per month) until the bugs are fixed. So the average 68 days should be a good approximation of the time for developers to actively diagnose and fix the bugs.

Chapter 7

Related Work

We are unaware of any prior work that analyzes textual bug reports to automatically generate patches.

7.1 Fault Repair

Many techniques fix a faulty program by modifying the program to satisfy the violated specifications or test cases. [7, 11, 14, 18, 41, 42]. In contrast to these techniques which require often unavailable specifications or test cases, R2Fix does not require any specifications or test cases. Recent techniques [23, 31] automatically patch atomicity violations. Sun et al. propagate bug fixes to all applicable locations [40]. Hussain et al. focus on the repair of complex data structures [20]. Lazaar et al. automatically correct constraint programs [28]. They did not analyze free-form bug reports to generate patches; some [23, 31] require the output of bug detection tools as input.

Dynamic and hybrid techniques [37, 39] are proposed to survive software failures and security attacks. While these techniques fix failures on a per execution basis, R2Fix is different because it (1) fixes the code for all future executions, and (2) fixes reported bugs described in English text. Users and developers will continue to report bugs despite the existence of these dynamic tools because (1) dynamic bug detection tools cannot detect all bugs; and (2) the monitoring and detection overhead is not justifiable for many software systems. R2Fix complements these techniques in improving software dependability. In addition, R2Fix is *safe* as patches are not applied until developers have confirmed the correctness of the patches.

7.2 Failure Diagnosis and Fault Localization

Many techniques find the root causes and other diagnostic information of software failures. LEAKPOINT [10] uses dynamic analysis to identify the last positions that leaked objects are alive, and then reset references to those objects to null at those positions to let objects be garbage collected. BugFix [22] leverages machine learning techniques to identify and report a prioritized list of suggestions about how to modify a statement to fix a bug. Wei et al. propose a framework that automatically generates interprocedural and path-sensitive analysis to detect faults specified by users [29]. HDD [34] is a hierarchical delta debugging approach with better scalability than previous delta debugging techniques to minimize failure-introducing inputs. Sarah et al. demonstrates that generating relatively short summaries for bug reports can help developers find duplicate bug reports more efficiently [38]. Mark Weiser studies programmers' behavior and finds that programmers usually breaks apart large programs into slices for the ease of debugging [43]. Xiangyu Zhang et al. design and evaluate different slicing algorithms [47]. Cristian Zamfir proposes a technique that automatically produces an execution replay of the given bug report and relevant program using a combination of static analysis and symbolic execution [46]. BugLocator [48] uses an information retrieval based approach to locate the places where the bugs should be fixed from given bug reports.

We take the diagnosis process one step further to directly generate patches with no developer involvement required. R2Fix complements previous work by leveraging free-form bug reports to help developers diagnose and fix bugs when run-time information and execution traces are unavailable.

7.3 Fix and Fix Pattern Studies

Compared to the fix pattern study by Pan et al. [36], our fix pattern study is finer-grained with a focus on the fix pattern semantics, so that our fix patterns can be used for specific bug fixing. In addition, we study C/C++ code fix patterns of different bug types separately, while the previous work analyzed fix patterns of different bug types aggregately for Java code. A few of our overflow fix patterns are from previous work [17]; this paper covers more bug types and more fix patterns. A recent study examines bug fixes that are cross-referenced between FreeBSD and OpenBSD [9].

7.4 Bug Report Prioritization

Philip J. Guo et al. perform an empirical study on factors that affect which bugs are fixed in Windows Vista and Windows 7. They also build a static model based on the study to predict the probability that a new introduced bug will be fixed [15]. While prioritizing which bugs to be fixed can potentially prompt a quick response to bug reports, developers still need to generate the patches themselves. With patches generated, R2Fix's results are actionable: developers can verify and apply the patches.

7.5 Bug Report Classification

Various techniques are proposed to identify memory and semantic bug reports [30], predict the bug lifetime and bug-fix time [13, 19], and bugs to be fixed [15]. Our classifiers address a different problem. The prior work [30] cannot accurately classify bug reports to fine-grained bug types, e.g., buffer overflows, because random sampling does not produce sufficient training data. R2Fix's bug classifiers address this issue by combining random sampling with keyword search.

Chapter 8

Conclusions

8.1 Conclusions and Future Work

This paper proposes a general approach, R2Fix, to automatically fix bugs by analyzing free-form bug reports. R2Fix has generated 60 *correct* patches with high precision, 5 of which are *new* patches generated for unfixed bug reports. We reported all 5 new patches, 4 have already been accepted and committed to the code repositories by the developers. Our detailed bug fix pattern study finds that a significant amount of buffer overflows, null pointer bugs, and memory leaks are fixed by a few simple fix patterns. We build classifiers to automatically identify these types of bug reports, which can be used for other purposes such as evaluating bug detection tools, studying the evolution of certain types of bugs, etc. In the future, we plan to generate patches for new types of bug reports, and extend R2Fix to take the output of existing bug detection tools as input to improve the effectiveness of patch generation. Given that patches share common fix patterns, we would like to build fix pattern databases and use them to guide future bug fixes.

APPENDICES

Appendix 1. Survey Example I

Dear Developer:

My name is Chen Liu, from University of Waterloo. We invite you to take a few minutes to answer three questions about the attached patch for one bug in the Mozilla Bugzilla database. The patch is generated by our automatic patch generation tool R2Fix. Your valuable feedback can help us understand the usability and benefits of patches automatically generated by R2Fix. R2Fix generated the patches by using only the information in the initial bug report, which only includes the "Summary/Title" field and the "Description" field of the bug. In other words, R2Fix does not use the "Comment" fields of the bug reports, because we want to apply R2Fix as soon as a bug is reported to maximize the time and effort that R2Fix can save for developers in fixing bugs.

The survey is attached at the end of this email in plain text format. The same survey is also attached in pdf format for your convenience. You can either respond in plain text or by modifying the pdf file. We estimate that it will take you approximately 3 minutes to complete this short survey. We would highly appreciate your response.

Your input is very important to us and will be kept strictly confidential. It will be used only for the purposes of research for this project.

Thank you very much for your time.

Best,
Chen Liu
University of Waterloo

P.S. Here is the survey:

R2Fix generates the following patch from bug report #201547 (https://bugzilla.mozilla.org/show_bug.cgi?id=201547):The patch fixes the reported buffer overflow bug by assigning a longer buffer.

Please read the patch and the bug report's "Summary/Title" and "Description" (do NOT read the "Comment" fields of the bug report), and answer the following questions.

```
--- a/mozilla/mozilla/mailnews/mime/src/mimetric.cpp
+++ b/mozilla/mozilla/mailnews/mime/src/mimetric.cpp
@@ -103,7 +103,7 @@ MimeRichtextConvert (char *line, PRInt32
const char *this_end;
int desired_size;
- desired_size = length * 4;
+ desired_size = (length * 5) + 1;
if (desired_size >= *obuffer_sizeP)
status = mime_GrowBuffer (desired_size, sizeof(char), 1024,
obufferP, obuffer_sizeP);
```

---

1. Would the automatically-generated patch save you time in diagnosing and fixing this bug?
__Yes
__No

1a. If yes, why? Mark all reasons that apply.
__Save my time in understanding the bug
__Save my time in retrieving the right version of the software and the files
__Save my time in fixing the bug and generating the patches using ”diff” or other commands
__Other, specified below:

1b. If no, why?

__The patch is simple enough for me to generate quickly myself.
__Other, specified below:

2.Would thee two automatically-generated patch prompt a quicker response to this bug report?
__Yes
__No

2a. If yes, why? Mark all reasons that apply.
__A bug report with patches attached is more likely to describe a valid bug.
__A bug report with patches attached is easier to fix.
__Other, please specify the reasons:

2b. If no, why? Please specify the reasons:

3. What concerns would you have in using an automatically generated patch?

Appendix 2. Survey Example II

Dear Developer:

My name is Chen Liu, from University of Waterloo. We invite you to take a few minutes to answer three questions about the attached patches for one bug in the kernel Bugzilla database. The patches are generated by our automatic patch generation tool R2Fix. Your valuable feedback can help us understand the usability and benefits of patches automatically generated by R2Fix. R2Fix generated the patches by using only the information in the initial bug report, which only includes the "Summary/Title" field and the "Description" field of the bug. In other words, R2Fix does not use the "Comment" fields of the bug reports, because we want to apply R2Fix as soon as a bug is reported to maximize the time and effort that R2Fix can save for developers in fixing bugs.

The survey is attached at the end of this email in plain text format. The same survey is also attached in pdf format for your convenience. You can either respond in plain text or by modifying the pdf file. We estimate that it will take you approximately 3 minutes to complete this short survey. We would highly appreciate your response.

Your input is very important to us and will be kept strictly confidential. It will be used only for the purposes of research for this project.

Thank you very much for your time.

Best,
Chen Liu
University of Waterloo

P.S. Here is the survey:

R2Fix generates the following two patches from bug report #13437 (https://bugzilla.kernel.org/show_bug.cgi?id=13437): The first patch fixes the reported buffer overflow bug by writing fewer bytes into buffer. R2Fix generates another patch for this bug report by allocating a longer buffer.

Please read the patches and the bug report's "Summary/Title" and "Description" (do NOT read the "Comment" fields of the bug report), and answer the following questions.

The first patch:

```
--- a/drivers/scsi/gdth_proc.c
+++ b/drivers/scsi/gdth_proc.c
@@ -151,6 +151,7 @@ static int gdth_set_asc_info(struct Scsi
 static int gdth_get_info(char *buffer,char **start,off_t offset,
 int length,struct Scsi_Host *host, gdth_ha_str *ha)
 {
+    int hlen = 0;
     int size = 0,len = 0;
     off_t begin = 0,pos = 0;
     int id, i, j, k, sec, flag;
@@ -192,11 +193,11 @@ static int gdth_get_info(char *buffer,ch
     if (reserve_list[0] == 0xff)
         strcpy(hrec, "--");
     else {
-        sprintf(hrec, "%d", reserve_list[0]);
+        hlen = sprintf(hrec, "%d", reserve_list[0]);
         for (i = 1;  i < MAX_RES_ARGS; i++) {
             if (reserve_list[i] == 0xff)
                 break;
-            sprintf(hrec,"%s,%d", hrec, reserve_list[i]);
+            hlen += snprintf(hrec + hlen, 161 - hlen, ",%d",
+                                    reserve_list[i]);
         }
     }
     size = sprintf(buffer+len,
```

The second patch:

```
--- a/drivers/scsi/gdth_proc.c
+++ b/drivers/scsi/gdth_proc.c
@@ -161,7 +161,7 @@ static int gdth_get_info(char *buffer,ch

     gdth_cmd_str *gdtcmd;
```

```
      gdth_evt_str *estr;
-     char hrec[161];
+     char hrec[322];
      struct timeval tv;

      char *buf;
```

---

1. Would the automatically-generated patches save you time in diagnosing and fixing this bug?
__Yes
__No

1a. If yes, why? Mark all reasons that apply.
__Save my time in understanding the bug
__Save my time in retrieving the right version of the software and the files
__Save my time in fixing the bug and generating the patches using "diff" or other commands
__Other, specified below:

1b. If no, why?

__The patches are simple enough for me to generate quickly myself.
__Other, specified below:

2.Would these two automatically-generated patches prompt a quicker response to this bug report?
__Yes
__No

2a. If yes, why? Mark all reasons that apply.
__A bug report with patches attached is more likely to describe a valid bug.
__A bug report with patches attached is easier to fix.
__Other, please specify the reasons:

2b. If no, why? Please specify the reasons:

3. What concerns would you have in using an automatically generated patch?

Appendix 3. Fix Pattern Table

Pattern	Subpattern	Param	Explanation
Overflow: LonferBuf	- TYPE BUF[OLD_LENGTH]; + TYPE BUF[NEW_LENGTH];	BUF [[& NEW_LENGTH]]	Declare the buffer with the new buffer length. If a new length is not found, simply double the old length.
	- LENGTHVAR = OLD_LENGTH; + LENGTHVAR = NEW_LENGTH;	LENGTHVAR & OLD_LENGTH [[& NEW_LENGTH]]	Assign the length variable a new length. If a new length is not found, simply double the old length.
FewerByte	- strcpy(BUF,EXPR); + strlcpy(BUF,EXPR, sizeof(BUF));	none	Replace strcpy function with strlcpy function.
	- sprintf(BUF,FMT,EXPR); + snprintf(BUF, sizeof(BUF),FMT,EXPR);	none	Replace sprintf function with snprintf function
	+ int hlen; ... - sprintf(BUF,FMT,EXPR) + hlen = snprintf(BUF, sizeof(BUF), + FMT,EXPR); [[... - sprintf(BUF,''%s FMT2'',BUF,EXPR2); + hlen += snprintf(BUF+hlen, + sizeof(BUF)-hlen,'' FMT2'',EXPR2);]]+	none	If the same buffer is written in different sprintf calls, we control the number of bytes written into the buffer.
MdBound	- if(EXPR [<|<=|>=|>] EXPR2) + if(EXPR [<=|<|>|>=] EXPR2)	[¡—¡=—¿=—¿]	Look for inequality such as if (a ¿ b) in the bug report. Modify the operation of the inequality in the if statement.
Nullptr: AddCheck	+ if (PTR) [[EXPR =]] FNC(...,PTR,...);	FNC & PTR	Add a null check before PTR is passed to FNC as a parameter.
	FNC(...){... +if(!PTR) return; EXPR1 = PTR->EXPR2; ...}	FNC & PTR	Add a null check before PTR is dereferenced.
	PTR = FNC(...); + if(!PTR) return NULL;	FNC & PTR	Add a null check after PTR is assigned the return value of FNC.
	if ([[EXPR||]]* + !PTR|| !FNC(...,PTR->FLD,...))	FNC or PTR	Add a null check for PTR in the 'if' statement before PTR is dereferenced.
	if ([[EXPR||]]* + PTR|| FNC(...,PTR,...))	FNC or PTR	Add a null check for PTR in the 'if' statement before PTR is passed as a parameter.

AddCheck	`if ([[EXPR &&]]*`
	`+ !PTR &&`
	`FNC(...,PTR,...))`
		FNC or PTR	Add a null check for PTR in the 'if' statement before PTR is passed as a parameter.

	`FNC(...){...`
	`+ if (PTR)`
	`PTR->FLD = EXPR;`
	`...}`
		FNC & PTR	Add a null check for PTR in FNC before PTR is dereferenced.

MvCheck	`- TYPE I = PTR->FLD;`
	`+ TYPE I;`
	`... if (!PTR) {...}`
	`+ I = PTR->FLD;`
		none	Move the null check of PTR before the derefenrence of PTR.

	`- PTR[I] = EXPR;`
	`... if (!PTR) {...}`
	`+ PTR[I] = EXPR;`
		none	Move null check of array PTR before accessing PTR

RmCheck	`TYPE I = PTR->FLD; ...`
	`- if (PTR){`
	`...`
	`- }`
		none	Remove the unnecessary null check after the dereference of PTR.

	`TYPE T = PTR->FLD; ...` `- if (!PTR) {...}`	none	Remove the unnecessary null check after the dereference of PTR.

Leak: AddFree	`TYPE *PTR = ...; ...`
	`if(EXPR)`
	`+ { delete PTR;`
	`return ...;`
	`+ }`
		none	Make sure that PTR is deleted when EXPR is evaluated to be true.

	`TYPE *BUF;`
	`...`
	`+ kfree(BUF)`
		LINE NUMBER	Free the allocated buffer at given line.

Table 8.1: R2Fix Appendix: Fix Patterns. [[]] represents optional part, and [[]]* represents this pattern could appear 0 or more times."Size" and "len" are often part of a buffer length variable name; once a buffer length variable is identified, the buffer length can be extracted simply by checking right hand side of "=". To extract the line number for memory leak bugs, R2Fix takes the number after ":" or "at line" in the bug report. Note that the line number is only required for one memory leak subpattern, and is not needed for any other subpatterns.

References

[1] Apache Bugzilla. https://issues.apache.org/bugzilla/.

[2] Cppcheck. http://cppcheck.sourceforge.net/.

[3] Linux Bugzilla. http://bugzilla.kernel.org/.

[4] Mozilla Bugzilla. https://bugzilla.mozilla.org/.

[5] National vulnerability database. http://nvd.nist.gov/.

[6] John Anvik, Lyndon Hiew, and Gail C. Murphy. Coping with an open bug repository. Eclipse'05.

[7] Andrea Arcuri. On the automation of fixing software bugs. In *ICSE'08 Companion*.

[8] Cristian Cadar, Daniel Dunbar, and Dawson R. Engler. Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs. In *OSDI*.

[9] Gerardo Canfora, Luigi Cerulo, Marta Cimitile, and Massimiliano Di Penta. Social interactions around cross-system bug fixings: the case of freebsd and openbsd. MSR'11.

[10] James A. Clause and Alessandro Orso. LEAKPOINT: Pinpointing the causes of memory leaks. In *ICSE'10*.

[11] Valentin Dallmeier, Andreas Zeller, and Bertrand Meyer. Generating fixes from object behavior anomalies. In *ASE'09*.

[12] Jon Eyolfson, Lin Tan, and Patrick Lam. Do time of day and developer experience affect commit bugginess? In *MSR'11*.

[13] Emanuel Giger, Martin Pinzger, and Harald C. Gall. Predicting the fix time of bugs. RSSE'10.

[14] Claire Le Goues, Michael Dewey-Vogt, Stephanie Forrest, and Westley Weimer. A systematic study of automated program repair: Fixing 55 out of 105 bugs for $8 each. ICSE'12.

[15] Philip J. Guo, Thomas Zimmermann, Nachiappan Nagappan, and Brendan Murphy. Characterizing and predicting which bugs get fixed. ICSE'10.

[16] Munawar Hafiz. *Security On Demand*. PhD thesis, 2010.

[17] Munawar Hafiz, Paul Adamczyk, and Ralph Johnson. Systematically eradicating data injection attacks using security-oriented program transformations. ESSoS'09.

[18] Haifeng He and Neelam Gupta. Automated debugging using path-based weakest preconditions. In *FASE'04*.

[19] Pieter Hooimeijer and Westley Weimer. Modeling bug report quality. ASE'07.

[20] Ishtiaque Hussain and Christoph Csallner. Dynamic symbolic data structure repair. In *ICSE'10*.

[21] Nicholas Jalbert, Cristiano Pereira, Gilles Pokam, and Koushik Sen. Radbench: a concurrency bug benchmark suite. HotPar'11.

[22] Dennis Jeffrey, Min Feng, Neelam Gupta, and Rajiv Gupta. Bugfix: A learning-based tool to assist developers in fixing bugs. In *ICPC'09*.

[23] Guoliang Jin, Linhai Song, Wei Zhang, Shan Lu, and Ben Liblit. Automated atomicity-violation fixing. In *PLDI'11*.

[24] Wei Jin and Alessandro Orso. Bugredux: Reproducing field failures for in-house debugging. In *ICSE'12*.

[25] Sunghun Kim and E. James Whitehead, Jr. How long did it take to fix bugs? In *MSR'06*.

[26] Ludmila I. Kuncheva and Sushmita Mitra. A two-level classification scheme trained by a fuzzy neural network. In *ICPR'94*.

[27] Thomas D. LaToza, Gina Venolia, and Robert DeLine. Maintaining mental models: A study of developer work habits. In *ICSE'06*.

[28] Nadjib Lazaar, Arnaud Gotlieb, and Yahia Lebbah. A framework for the automatic correction of constraint programs. In *ICST'11*.

[29] Wei Le and Mary Lou Soffa. Generating analyses for detecting faults in path segments. ISSTA'11.

[30] Zhenmin Li, Lin Tan, Xuanhui Wang, Shan Lu, Yuanyuan Zhou, and Chengxiang Zhai. Have things changed now? – An empirical study of bug characteristics in modern open source software. In *ASID'06*.

[31] Peng Liu and Charles Zhang. Axis: Automatically fixing atomicity violations through solving control constraints. ICSE'12.

[32] Shan Lu, Soyeon Park, Eunsoo Seo, and Yuanyuan Zhou. Learning from mistakes — A comprehensive study on real world concurrency bug characteristics. In *ASPLOS'08*.

[33] Christopher D. Manning and Hinrich Schütze. *Foundations Of Statistical Natural Language Processing*. The MIT Press, 2001.

[34] Ghassan Misherghi and Zhendong Su. HDD: Hierarchical delta debugging. In *ICSE'06*.

[35] Yoann Padioleau, Julia L. Lawall, and Gilles Muller. SmPL: A domain-specific language for specifying collateral evolutions in linux device drivers. *ENTCS'07*.

[36] Kai Pan, Sunghun Kim, and E. James Whitehead. Toward an understanding of bug fix patterns. *ESE'09*.

[37] Jeff H. Perkins, Sunghun Kim, Samuel Larsen, Saman P. Amarasinghe, Jonathan Bachrach, Michael Carbin, Carlos Pacheco, Frank Sherwood, Stelios Sidiroglou, Greg Sullivan, Weng-Fai Wong, Yoav Zibin, Michael D. Ernst, and Martin C. Rinard. Automatically patching errors in deployed software. In *SOSP'09*.

[38] Sarah Rastkar, Gail C. Murphy, and Gabriel Murray. Summarizing software artifacts: a case study of bug reports. In *ICSE'10*.

[39] Jocelyn Simmonds, Shoham Ben-David, and Marsha Chechik. Guided recovery for web service applications. In *FSE'10*.

[40] Boya Sun, Gang Shu, Andy Podgurski, Shirong Li, Shijie Zhang, and Jiong Yang. Propagating bug fixed with fast subgraph matching. In *ISSRE'10*.

[41] Westley Weimer. Patches as better bug reports. In *GPCE'06*.

[42] Westley Weimer, ThanhVu Nguyen, Claire Le Goues, and Stephanie Forrest. Automatically finding patches using genetic programming. In *ICSE'09*.

[43] Mark Weiser. Programmers use slices when debugging. *Communications of the ACM*, 1982.

[44] Ian H. Witten and Eibe Frank. *Data Mining: Practical machine learning tools and techniques (2nd Ed.).* Morgan Kaufmann, 2005.

[45] Zuoning Yin, Ding Yuan, Yuanyuan Zhou, Shankar Pasupathy, and Lakshmi. How do fixes become bugs? – A comprehensive characteristic study on incorrect fixes in commercial and open source operating systems. In *FSE'11*.

[46] Cristian Zamfir and George Candea. Execution synthesis: A technique for automated software debugging. In *Eurosys'10*.

[47] Xiangyu Zhang, Rajiv Gupta, and Youtao Zhang. Precise dynamic slicing algorithms. In *ICSE'03*.

[48] Jian Zhou, Hongyu Zhang, and David Lo. Where should the bugs be fixed? - More accurate information retrieval-based bug localization based on bug reports. ICSE'12.