Aritra Dhar <aritra1204@iiitd.ac.in>

ESEC/FSE 2015: Acceptance Nntification for paper number 281

1 message

ESEC/FSE 2015 <esecfse2015@easychair.org> Tue, May 26, 2015 at 11:01 PM
To: Aritra Dhar <aritra1204@iiitd.ac.in>

Dear Aritra,

Thank you for your submission to the ESEC/FSE 2015 conference, entitled:

Clotho:Saving Programs from Malformed Strings and Incorrect String-handling

The program committee met on May 23rd and 24th 2015 in Florence to discuss the submitted papers. We are happy to inform you that your submission was accepted for inclusion in the ESEC/FSE 2015 conference program. Many congratulations!

Each paper went through a thorough review process. Overall, 74 out of 291 submissions were accepted (a 25.4% acceptance rate).

The reviews of your paper are enclosed. Please make sure to carefully take into account the comments of the reviewers when preparing the camera-ready version of your paper. It is important that you address all the revisions requested by the referees for the final, camera-ready copy of the paper.

The camera ready copy is due by July 15th, 2015. Just like your submission, the final version of your paper must not exceed 10 pages plus a reasonable number of additional pages consisting solely of references. The publisher will follow up with more information soon. Please expect an author-kit to be sent in a separate email. The procedure used by the publishers is explained here: https://www.conference-publishing.com/Procedure.html

A note on replication packages: Results presented in technical papers are often validated or supported by software artifacts, including tools, models, and datasets. To reward the effort of creating these artifacts and enhance reproducibility of results presented at the conference, all the authors of accepted full papers, such as yourselves, have the opportunity to additionally submit their artifacts to the Replication Packages Evaluation Committee. Papers associated with accepted replication packages will have five extra minutes of presentation time at the conference, an extra page dedicated to the presentation of the replication package in the conference proceedings, and the replication package itself stored in the ACM digital library. The deadline for submission is June 7. For more information, please check the website: http://esec-fse15.dei.polimi.it/replicationPack.html

Thank you, once again, for your paper. We look forward to seeing you at ESEC/FSE 2015 in Bergamo!

With very best wishes and congratulations,

Mark Harman and Patrick Heymans,
ESEC/FSE 2015 Program Co-chairs.

---------- REVIEW 1 ----------
PAPER: 281
TITLE: Clotho:Saving Programs from Malformed Strings and Incorrect String-handling

AUTHORS: Aritra Dhar, Rahul Purandare, Mohan Dhawan and Suresh Rangaswamy

---------- SUMMARY AND EVALUATION ----------
Summary

This paper presents an approach to repair Java programs that would throw exceptions due to malformed strings or incorrect handling of strings. The authors justify their choice to focus on String-related exceptions by two arguments: (a) frequent usage of strings in Java software, especially Java libraries, and frequency of exceptions related to Strings in such software, and (b) the advantages of targeting a specific data structure with a well defined API in terms of being able to exploit domain knowledge to precisely identify the points in code where to inject patches as well as the ability to automatically synthesize precise patches that not only prevent the program from crashing but do so without effecting the intended behavior.

The paper presents the design and implementation of CLOTHO, a tool for repairing Java software that incorrectly uses Strings. CLOTHO leverages a combination of program analysis techniques to identify places to repair.
It exploits intra-procedural static and dynamic analysis to identify and evaluate constraints on the string objects under consideration and uses these constraints to generate patches that are embedded into try-catch blocks to ensure that they only become active, when exceptions are actually thrown at runtime.

The proposed approach is evaluated by applying it to several open-source libraries to patch 30 bugs from respective bug repositories resulting from unhandled runtime exceptions from Java String API. The authors characterize the effectiveness of CLOTHO to patch the bugs based on the number of passing tests and judge the quality of the generated patches by comparing them with developer patches.

Evaluation

The paper addresses a focussed but relevant problem. The proposed technique seems reasonable although unfortunately not very well presented (see below). The evaluation targets real-world popular libraries and real bugs; the experimental setup and the evaluation methodology are appropriate and well designed.

The main problem with the paper concerns the presentation of the approach and its evaluation.

Concerning the presentation of the approach, some key design decisions are not properly justified. This is especially true for the decision to exclude from the repair process program statements that lie along a control flow path that leads to an I/O sink. A taint analysis is performed to identify string objects that lie along paths from sensitive sources to sensitive sinks. These objects are marked as unsafe for repair. This reviewer does not understand why one would want this. On the contrary, it seems that one would want to repair critical paths, e.g., to prevent security exploits. Furthermore, it seems that the taint analysis is not performed in the benchmarks, arguing that since the latter are libraries, they do not have a notion of sources and sinks. Doesn't this basically mean that the taint analysis is worthless anyway? So, why is it featured?

The constraint set that CLOTHO uses for patch generation should be explained in more detail. How is the set derived? Why is it sufficient? BTW, Fig. 2 that shows the set is almost impossible to read. Terms are used without being introduced. E.g., the term "declarative constraint" is used in the beginning of the paragraph on "static constraint collection" without being defined. What are "declarative constraints"? The use of the term suggest that there are also "non-declarative constraints". What are those?

Overall the description of the static collection of constraints is vague. The author just refer to the algorithm 3 for a definition of the semantics, without further explanation. This reviewer fails to understand how "preferred branches" are determined and how they actually contribute to the collection of constraints. It seems that the authors perform some data flow analysis, but other than that the details remain obscure. An example would probably help a lot to provide the reader with an intuition of how static constraint collection works.

Similar remarks apply to the dynamic collection of constraints. This reviewer fails to understand when the dynamic collection is necessary and how it works. In this case, the authors give an example, but its description does not help

much.

In both cases, how are the constraints evaluated and how does their evaluation affects patch generation? The presentation of the object repairing in page 5 is as vague as the constraint generation. Consider e.g., the following statement: “If the constraints are resolved statically, then CLOTHO updates its constraint data store (BTW: This may be the first time this term is used without being defined) and instruments the corresponding byte codes appropriately.” “Doing something appropriately” does not explain how the thing is done in the first place to allow to reader to judge whether this is appropriate or not.

Concerning the presentation of the evaluation, here are some questions that came up during reading. The terms “forced patching” was used without being defined before - at least, if there was a definition, this seems to be not explicit (prominent) enough and has skipped the attention of this reviewer. What non-String related cascaded exceptions were thrown by the patched versions of CLI2.x Eclipse AJ Weaver? Wouldn’t it be more reasonable to exclude such failures from the consideration in the first place, as they are not in scope of the approach? Concerning the calculation of PPI, “… thereby considering the core logic to construct an effective patch.” What does that mean? The authors explain that both the number constraints in the generated patches as well as those in the developer’s patch can be automatically calculated. But, they calculate the latter manually. Why? What does that mean for the precision of PPI? Concerning the explanation for benchmarks with PPI! < 0.7, it is strange that the authors didn’t exclude data types other than strings during the manual calculation of PPI. This seems to suggest that the calculation was done automatically and not very precisely?

Overall, the paper could profit from being less verbose in some parts and more detailed and more precise in other parts. For example a significant part of the introduction - up to the beginning of page 2 - can be skipped and the authors could more directly start with the concrete problem they address.

---------- STRENGTHS AND WEAKNESSES ----------
Strengths:

+ relevant problem, clear motivation for targeting the specific problem
+ well designed experimental setup
+ reported results provide evidence for the claims

Weaknesses:

- targeted problem is rather narrow. the authors claim that the approach can generalize to other kinds of bugs by exchanging the constraint resolution strategy and by providing other formal specifications of the API behavior, but there is evidence for this

- presentation of the approach and of the evaluation is vague and imprecise

---------- REVIEW 2 ----------
PAPER: 281
TITLE: Clotho:Saving Programs from Malformed Strings and Incorrect String-handling
AUTHORS: Aritra Dhar, Rahul Purandare, Mohan Dhawan and Suresh Rangaswamy

---------- SUMMARY AND EVALUATION ----------
SUMMARY

This paper introduces a hybrid static and dynamic approach to recover programs from crashes due to malformed or inappropriate handling of strings. The technique operates in three stages: (1) statically identifying string objects or API arguments to be repaired for preventing runtime exceptions; (2) performing intra-procedural static and dynamic analyses to collect and evaluate constraints on identified string objects; and (3) using the evaluated constraints to generate and deploy patches within catch blocks to avoid affecting the normal

control-flow.

The technique has been implemented in the CLOTHO tool for the JAVA String API, and applied to popular open-source libraries to patch 30 bugs. The results show that the tool is effective. The tool and data have also been made available.

EVALUATION

Overall, this is an interesting piece of work. It tackles the problem of automatic recovery for a specific, but important problem. The general approach is well developed. Although I feel that none of the analysis components stands out on its own, the integrated overall system is indeed quite solid and effective. The evaluation has been carefully conducted with convincing results and useful details. The paper itself is also generally well written (see comments below for some suggestions).

I would like to point out two weaknesses that I hope the authors can consider and possible address. One concerns the somewhat narrow focus of the work and technique. This is okay, but I think it would help the reader if the authors could discuss whether there is anything general that can be leveraged in the work to deal with non-string related failures. Some examples and discussions would make the work more complete and informative.

The second concerns the Section 4's technical presentation. The high-level ideas are sufficiently clear and understandable, but unfortunately I also feel that many of the details are not explained well enough and feel vague. One plausible way to improve the presentation is to use a completely worked out example to illustrate the various aspects of CLOTHO, so that the reader could gain a better, clearer understanding. The authors are strongly encouraged to do so. If possible, a more formal, precise presentation would also be more desirable. As an example, the paper never makes clear the constraint language.

More comments:

- p2,c1: regarding the study on stackoverflow posts where 33K out of 60K on Java exceptions were found to do with string objects.

This is an interesting analysis, but perhaps it would be better to perform similar analysis on bugs in the bug trackers of popular projects.

- p2,c2: "Ensuring correctness of a program statically is an undecidable problem. However, there is always a tradeoff between precision and scalability ...": However => Thus

- p2,c2: by making sound approximations: does not need to be sound

- p2,c2: "... leads to several false positives": delete "several"

- p3,c1: "may [be] used to print information on screen"

- p3,c1: the subscripts get a bit ugly; consider typeset them differently or better

- p3,c1: "is close to the intended behavior": How would you know what is the intended behavior? It is easier to only avoid runtime exceptions, which is what is done in the work.

- p6,c2: "performs [a] few ..."

- p7,c1: "We mined bug repositories of several open-source JAVA-based applications and selected 30 bugs": Please clarify how the 30 bugs were selected. Randomly? Based on importance, etc.?

- p7,c1: May'14 => May 2014

- p7,c1: "against the the benchmark's ...": extra "the"

- Consider swapping Tables 4 (having it on page 8) and 3 (having it on page 7) so that they are in order and also closer to their respective textual descriptions.

- p9,c1: "and not in the was actually not in the Aries framework as originally": fix

---------- STRENGTHS AND WEAKNESSES ----------
+ interesting, important problem
+ effective hybrid approach for repair
+ detailed evaluation on real-world failures
- limited to String related issues
- techniques do not seem generally applicable
- paper quite well written (but the core part can be presented clearer)

---------- REVIEW 3 ----------
PAPER: 281
TITLE: Clotho:Saving Programs from Malformed Strings and Incorrect String-handling
AUTHORS: Aritra Dhar, Rahul Purandare, Mohan Dhawan and Suresh Rangaswamy

---------- SUMMARY AND EVALUATION ----------
This submission presents Clotho, an automatic program repair approach that focuses on repairing string handling errors. Clotho identifies string operations in a program that it can patch while preserving 1) the program's critical value at a given string operation and 2) keeping the rest of the program's critical "close to the intended behavior". To preserve these properties, they conservatively identify program points where they can apply their patches. They use taint analysis to identify strings involved in environmental interactions, either as reads or writes. An exception may not be handled current function, but instead by a function in its call stack. They analyze the call graph to find upstream handling of string exceptions, and eliminate, as instrumentation targets, any string operations dominated by such exception handling. As optimization, they do not instrument string operations dominated by previous patched operations. Having identified target string operations, Clotho harvests constraints on the string arguments statically and dynamically, uses these constraints to patch the operations by wrapping the string operations in try catch blocks, whose blocks execute the repair operation if the constraints are met.

I agree with the author's contention that their focus on string operations is a strength of their approach. I believe operation targeted repair is a promising research direction and that string operations are a natural and important target.

In Section 3, the second constraint of your problem formalization says that "the resulting behaviour $\mathcal{B_{I_{f_n}}}'$ is close to the intended behaviour $\mathcal{B_{I_{f_n}}}$". What does "close to" mean? What about $\mathcal{B_{I_{f_i}}}$", for $i \ne n$?

Externally visible program behavior is not necessarily critical, but you equate the two. Your work would be strengthened by justifying equating them.

The syntax of conditional expression you can handle is unclear. It's implicit to your algorithms. For instance, can you handle disjunctive string constraints? What about string constraints that involve variables in conjuncts that do not include string operations, like x > 10 in s.length() < x ^ x > 10.

Evaluation

How did you select the 30 bugs you collected? Stating your selection criteria allows the reader to evaluate Clotho's general applicability.

You use taint, control flow, and reachability analysis to eliminate string operations as instrumentation points. In 4.2 (1), you ignore branch that throw exceptions or error paths. This yet another pruning heuristic. After all the pruning of instrumentation points, can you quantify how many instrumentation points remain?

Your PPI measure depends on some manual assessment similarity. Could you please detail the procedure you used to determine similarity?

Minor

Section 2:

The following the sentence should be rewritten:

The only exception that this invoke statement can throw is when the receiver object referenced by filename is null.

The code listings are very faint and hard to read when the paper is printed in black and white.

---------- STRENGTHS AND WEAKNESSES ----------
+ Domain-specific program repair, especially for such an important domain as String operations, is a promising research direction, as this work shows

- The paper's formalism needs refinement before this work will be ready for publication

- The evaluation does not address important questions about the technique, like its generality and how many instrumentation points remains after all of the filtering Clotho applies. Some of the measurement depend on human assessment that is unexplained.