RESEARCH ARTICLE OPEN ACCESS

Reducing Bug Triage in Software Development in Data Reduction

M. Ahilandesware [1], J. Prathiba [2]
PG scholar [1], Assistant Professor [2] Sr. Grade
Department of Computer Science and Engineering
SRM University, Chennai
Tamil Nadu -India

ABSTRACT

This Paper presents automatic bug triage. The algorithm for implementing the automatic bug triage is discussed. The advantage of implementing a bug triage is to minimize time and cost of manual work. The steps to identify the bug and to fix the bug are saved in the database. The above steps are automated to implement automatic bug triage. The same bug which was occurred previously repeatedly occurs next time, the system checks the database and the bug will be triaged and fixed using the saved steps. These results in reducing the data .Using this algorithm data replacement will be reduced. The Performance measurement will show the comparison between the previous report and reduced bug report.

Keywords:- Bug report, automatic bug triage, optimal asymmetric encryption padding, padding, Un-padding.

I. INTRODUCTION

Bug triage is meant for minimizing time and cost in software development. Bug triage aims to assign bugs to a developer who is capable of fixing bugs. Due to large number of bugs occurring and due to the lack of knowledge in fixing the bugs, Manual bug triage is expensive in cost and time. The time and cost between opening a bug and triaging a bug is 19.3 days. To reduce the time spent on triaging, we present an approach to semi automating one part of the process, the assignment of developer to a new bug report. This information can help the triage process in two ways: it may allow a triage to process a bug more quickly, and it may allow triages with less overall knowledge of the system to perform bug assignments more correctly as illustrated in Fig 1.

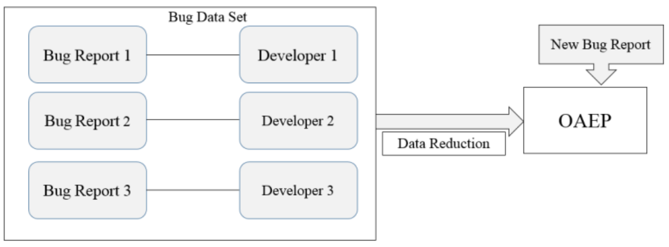


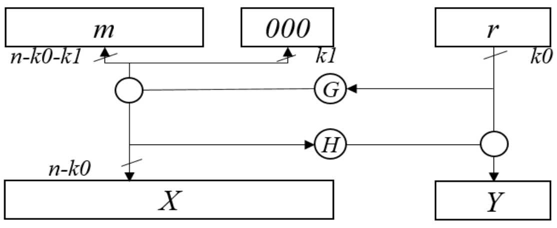
Fig 1 Architecture of Data Reduction

We use optimal asymmetric encryption padding with the SHA-256 and MFG1 Padding for data replacement. Two challenges which affect the use of repositories in software development tasks, the large scale and low quality. Existing work has proposed an automatic bug triage, which applies text classification techniques to map the bug report to the developer based on the results of text classification human triage assigns new bugs by his expertise.

Anvik reports that average of 37 bugs per day is submitted to the BTS and person on hours is required for manual bug triage. 44 % of bugs are assigned to the wrong developer.to solve those problems, use machine learning algorithm for automatic bug triage. We use machine learning algorithm for assigning bugs to the correct developer

II. ALGORITHM

A java implementation of optimal asymmetric encryption padding to be used in conjunction with RSA. Also includes MFG1 implementation. Optimal Asymmetric encryption padding is a padding scheme together with RSA encryption.



OAEP was introduced by Bellare and Rogway.

Fig 2 OAEP

The OAEP algorithm is a form of Feistel network which uses a pair of random oracles G and H to process the plaintext prior to asymmetric encryption as in Fig 2. When combined with any secure trapdoor one-way permutation F, this processing is proved in the random oracle model to result in a combined scheme which is semantically secure under chosen plaintext attack. When implemented with certain trapdoor permutations (e.g., RSA), OAEP is also proved secure against chosen cipher text attack. OAEP can be used to build an all-or-nothing transform

III. PADDING OF BYTES

byte[] pad(byte[] message, String params, int length);

It Returns a padded version of the message. params is a tokenizable String separated by spaces. The first argument should be the name of the hash function used, and the second argument should be the name of the mask function used. An example of a correct params string is SHA-256 MGF1, which also happens to be the only combination I have supported so far. Length is simply the desired final length of the message. If you were using RSA-2048, this value would be 256.

IV. UNPADDING OF BYTES

byte[] unpad(byte[] message, String params);

It Returns an unpadded version of the message given the params (these parameters should be the same as those used in the pad method).

byte[] MGF1(byte[] seed, int seed Offset, int seed Length, int desired Length);

In this paper I have also included an implementation of MGF1 with this code. It takes an input seed and an offset (seed Offset) and length (seed Length) so that only a slice of the seed may be used to generate masks. Desired Length is the length of the output, a mask generated from the given seed.

V. MFG1 MASKING

MGF1 is what I used for generating masks we are able to expand and shrink data using hash functions. Optimal Asymmetric Encryption Padding with SHA-256 message digest and MGF1 mask generation function. Padding schemes can be used in both asymmetric key ciphers as well as symmetric key ciphers

Block-ciphers especially regularly use padding schemes as they are based on the notion of fixed-length block sizes.

Public static final Padding Scheme OAEP with Sha512 and Mgf1

Optimal Asymmetric Encryption Padding with SHA-512 message digest and MGF1 mask generation function. This is a convenient pre-defined OAEP padding scheme that embeds the message digest and mask generation function. When using this padding scheme, there is no need to in it the Cipher instance with an OAEP Parameter Spec object, as it is already 'built in' to the scheme name

VI. MASK GENERATION FUNCTION

PSS requires a so-called mask generation function. This is basically like a hash function, but with a variable output size. In other contexts, similar functions are also called key derivation functions. The PKCS #1 v2.1 standard lists only one possible function, MGF1. It is based on an existing hash algorithm and just works by using the input plus a four byte counter starting with zero as an input for the hash function and increment the counter to get enough output bits from the hash function. The last output is cut to get the required size. MGF1 is mostly equivalent to the key derivation function KDF2,

MGF1 has the property that two calls to MGF1 with the same hash function and the same input with a different output size would lead to an output identical at the beginning. For example, if we calculate both MGF1(SHA-256, "hello", 10) and MGF1(SHA-256, "hello", 20), we get: MGF1(SHA-256, "hello", 10) = da75447e22f9f99e1be0 MGF1(SHA-256, "hello", 15) = da75447e22f9f99e1be09a00cf1a07

As we see, the first 10 bytes of the second MGF1 output are identical to the first MGF1 output.

VII. RESULTS AND OBJECTIVES

The data reduction can be shown using an X-graph result. XGRAPH is a general purpose x-y data plotter with interactive buttons for panning, zooming, printing, and selecting display options. It will plot data from any number of files on the same graph and can handle unlimited data-set sizes and any number of data files.

XGRAPH produces WYSIWYG PostScript, PDF, PPTX, and ODP output for printing hard-copies, storing, and/or sharing plotted results, and for importing, graphs directly into word-processors for creating documentation, reports, and view-graphs.

Data file formats: XGRAPH expects data in an x y format. Typically, this is one x-y data-point pair per line. Data values may be separated by white-space (spaces or tabs), commas, semi-colons, or colons. Multi-column data has several values per line. Each value, or column, is separated by white-space (spaces or tabs), commas, semi-colons, or colons. Any column can be selected as the ordinate or the abscissa by the '-c' column option.

The Graph shows the comparison between the previous and reduced bug report using X-graph as in Fig 3 and Fig 4. Data replacement time has been reduced using this algorithm optimal asymmetric encryption padding.

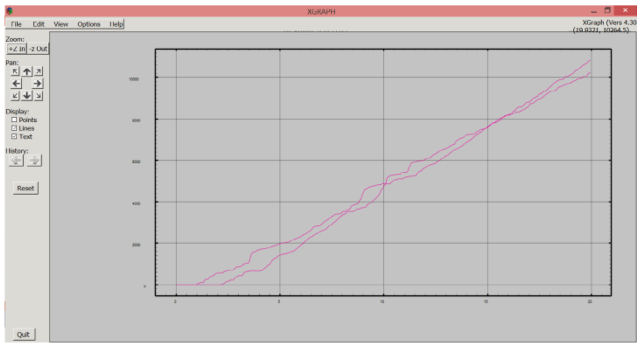


Fig 3 Before Data Reduction

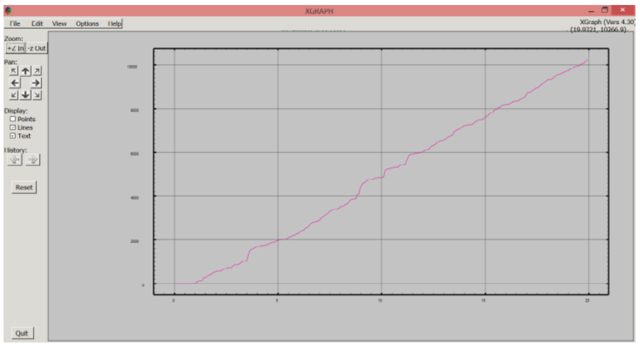


Fig 4: After Data Reduction

VIII. CONCLUSION

Manual bug triage is an expensive step in software development in the aspect of time and cost. In this paper, we use an optimal asymmetric encryption padding for encrypting and decrypting the data and to reduce the scale of bug data sets and to improve the quality of data. We use SHA-256 and MFG 1 padding with OAEP for padding and unpadding bytes. We empirically investigate the data reduction on many applications. This work provides an approach to techniques on processing of data in to reduced form and high –quality bug data in software development and maintenance. Using this approach, data replacement have been reduced.

In future work, we plan on improving the results of data reduction in bug triage to explore how to prepare a high quality bug data set and tackle a domain –specific software task. We plan to work on strengthening the security of this algorithm.

REFERENCES

[1] J. Anvik, L. Hiew, and G. C. Murphy, *Who should fix this bug? in Proc.* 28th Int. Conf. Softw. Eng., May 2.

[2] J. Anvik and G. C. Murphy, *Reducing the effort of bug report triage: Recommenders for development-oriented decisions,* ACM Trans. Soft. Eng. Methodol., vol. 20, no. 3, article 10, Aug. 2011.

[3] S. Artzi, A. Kie_zun, J. Dolby, F. Tip, D. Dig, A. Paradkar, and M. D.Ernst, *Finding bugs in web applications using dynamic test generation and explicit-state model checking, IEEE Softw.,* vol. 36,no. 4, pp. 474–494, Jul./Aug. 2010.

[4] C. C. Aggarwal and P. Zhao, *Towards graphical models for text processing,* Knowl. Inform. Syst., vol. 36, no. 1, pp. 1–21, 2013.

[5] D. _Cubrani_c and G. C. Murphy, *Automatic bug triage using text categorization,* in Proc. 16th Int. Conf. Softw. Eng. Knowl. Eng.,Jun. 2004, pp. 92–97.

[6] G. Jeong, S. Kim, and T. Zimmermann, *Improving bug triage with bug tossing graghs,* Proc. Joint Meeting European Software Engineering Conf. & ACM SIGSOFT Symp. Foundations.

[7] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun, *An approach to detecting duplicate bug reports using natural language and execution information,* in Proceedings of the 30th international conference on Software engineering, ser. *ICSE '08*. New York, NY, USA: ACM, 2008, pp. 461–470. [Online]. Available: http://doi.acm.org/10.1145/1368088.1368151.

[8] N. Okazaki and J. Tsujii, *Simple and efficient algorithm for approximate dictionary matching,* in Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010), Beijing, China, August 2010, pp. 851–859. [Online]. Available: http://www.aclweb.org/anthology/C10-1096.

[9] O. Baysal, M. W. Godfrey, and R. Cohen, *A bug you like: A framework for automated assignment of bugs,*

in Proceedings of the 17th IEEE International Conference on Program Comprehension (ICPC), 2009.

[10] F. Servant and J. A. Jones, *Whosefault: automatic developer-to-fault assignment through fault localization,* in Proceedings of the 2012 International Conference on Software Engineering, ser. ICSE 2012. Piscataway, NJ, USA: *IEEE Press, 2012*, pp. 36-46.Online]. Available:http://dl.acm.org/citation.cfm?id=2337223. 2337228.

[11] M. Bellare and P. Rogaway. *Optimal asymmetric encryption*. In Advances in Cryptology |Eurocrypt '94, pages 92 111, 1994.

[12] E. Fujisaki, T. Okamoto, D. Pointcheval, and J. Stern. *RSA OAEP is Secure under the RSA Assumption*. In Crypto '2001, LNCS 2139, pages 260{274. Springer-Verlag, Berlin, 2001.