Fuzzing: Testing Security in Maintenance Projects

Frank Simon, Daniel Simon
SQS Software Quality Systems AG, Stollwerckstraße 11, 51149 Cologne, Germany
Email: frank.simon|daniel.simon@sqs.com

Abstract: New trends in IT industry impose increasingly requirements on openness and interoperability via networks to enterprise software systems. As a consequence, more and more legacy applications are made available via interfaces more openly through mobile and insecure networks, thereby inducing security risks the initial designs have never had to account for. In this paper, we show how a highly automatable black-box method called *fuzzing* for testing security can be integrated into testing processes to increase interfaces of legacy application in terms of security profiles.

1 Introduction

Several IT technology trends (SOA, cloud computing, and the ever present mobility) contribute consciously to increasing networking readiness of applications. Following the new trends, everyone is expecting legacy applications to be accessible via mobile and internet connections rather than closed and secure enterprise intranets. At the same time, more and more business critical data are made available through such channels and as such increase business risks with regards to security. Long-living software systems and application never designed for the access via unsecured and open networks now have to be made ready to interact and interoperate through channels unforeseen at the time of initial development. To this end, we elaborate in this paper how the widely-known ideas of *fuzzing* can be integrated into standard test processes in order to increase the security of legacy applications that have undergone extension by many new interfaces.

2 Testing and Security

According to ISTQB [1], testing is *"The process consisting of all life cycle activities, both static and dynamic, concerned with planning, preparation and evaluation of software products and related work products to determine that they satisfy specified requirements, to demonstrate that they are fit for purpose and to detect defects."*

Testing is not only mandatory for new IT systems. Whenever a change to a system is implemented (e.g. by adding interfaces) it is the tester's task to assure "correctness" of the implementation of the change. This includes regression testing to demonstrate unchanged requirements' stability as well as testing new requirements. New requirements can induce adjustments for the regression testing as well: The requirement to open an existing IT system for mobile communication – as example – has not only to be tested for its own but might motivate deeper testing of directly connected components.

For a more systematic view on these implicit testing adjustments testing can be refined into four steps (a more general approach can be found in [2]:

1. Identification of test objects (What artefacts relevant for project success?)
2. Identification of quality attributes (What properties should the artefacts have?)
3. Determination of corresponding test activities to ensure artefacts having particular attributes
4. Clustering of test activities into test stages that can be executed in conjunction

This paper focuses the following aspect: Adding new interfaces creates new test objects as well as it produces new or at least adjusted priorities for quality attributes requiring additional test activities on all test stages. Quality attributes for software can be taken from ISO 25000 family of standards. [3] In particular when adding new service interfaces to legacy applications the first time, security should be seen as one of the top priorities. Security is defined in the ISO 25010 standard as the *Degree to which a product or system protects information and data so that persons or other products or systems have the degree of data access appropriate to their types and levels of authorization.*

3 Fuzzing of Software Interfaces

Fuzzing was developed at the University of Wisconsin in 1989 [4], [5]. Takanen et al [6] define fuzzing as follows:

„A highly automated testing technique that covers numerous boundary cases using invalid data (from files, network protocols, API calls, and other targets) as application input to better ensure the absence of exploitable vulnerabilities. The name comes from modem applications' tendency to fail due to random input caused by line noise on `fuzzy` telephone lines."

From the tester's perspective, fuzzing is

- a black-box test for interfaces as test objects, as it does not require knowledge of the underlying implementation;
- a test method for the quality attribute security, as it tries to identify errors in a system that compromise confidentiality, integrity, and availability;
- a negative test method, as it does not try to verify expected system behaviour.

- a brute force method, as it makes use of excessive, sometimes random, test data to exploit the interfaces;
- a boundary test, as it derives test data from specified valid input data and bombards the interfaces accordingly;
- an automated test method, as its use of mass data can only be deployed effectively driven by a machine.

As legacy systems are migrated into Service Oriented Architectures and open their interfaces to the "outside" world, in some cases through the provisioning of web service interfaces, fuzzing is gaining more and more relevance. System owners have started to realise the risks associated with widely (and sometimes through uncontrolled networks) accessible interfaces and the need to make those interfaces "bullet proof" also with regards to security aspects.

To date however, a systematic or holistic approach towards fuzzing has not been observed. Surprisingly, both modern practise oriented process models (e.g., Microsoft Development Lifecycle Model) mention as well as established standards such as ISTQB/ISEB [1] give little attendance to security tests and fuzzing so far.

To utilise fuzzing for legacy systems undergoing some enhancements by adding new interfaces the following can be stated as basis:

1. regression testing and testing the new requirements is done.
2. security as new attribute must be revisited and considered adequately as it might not have been in the legacy system's original setup and it has significant influence on a wide range of additional test objects as well.

In the following, a generic approach is drafted to integrate fuzzing into a generic test process covering a wide range of project types in a way to easily account for new security requirements.

4 Integration of fuzzing into the standard test process

In order to make use of fuzzing in software maintenance projects, we have to integrate the fuzzing methodology in the test process to link it with statement (1) mentioned above. However, in practice there exist several different standard test processes like ISTQB fundamental test process [7], TMap process [8], SCRUM [9], and ISO 29119 [10]). For a generic fuzzing integration these different processes were analysed and a generic test process for testing was derived. This meta process was designed as to define the integration points for fuzzing and make available the fuzzing methods for a wide range of software projects.

Traditional test processes applied for retesting legacy systems can be easily aligned with this generic process. The artefacts in red define the integration points of fuzzing.

The advantages of this approach are:

- Existing test processes remain unchanged (or even better: a generic process is modelled that can be reused).
- Hotspots for integrating fuzzing are highlighted.
- This process complements "traditional" retesting with security testing

One point is not solved by this: The overall awareness that security gets a more important quality attribute motivating to apply this process. This creation of awareness has to be done separately in advance of deploying an IT-system.

5 Summary and Outlook

With the first results, we have integrated *fuzzing* into generic test processes and thereby have made available a method for security testing for general use in software development and maintenance. Fuzzing has already proven its success in many projects, however without any opportunity to report about tangible figures. Future work lies in a scientific case study, collecting relevant numbers (effort, findings, etc.) to demonstrate the positive ROI of this overall concept.

6 Acknowledgements

Parts of this work have been sponsored by SesamBB: *Security and Safety made in Berlin-Brandenburg e.V.* The full result is available to SesamBB members.

7 References

[1] Tilo Linz Andreas Spillner, *Basiswissen Softwaretest: Aus- und Weiterbildung zum Certified Tester - Foundation Level nach ISTQB-Standard*, 4th ed.: dPunkt Verlag, 2010.

[2] Frank Simon and Daniel Simon, *Qualitätsrisikomangement*. Berlin: Logos Verlag, 2010.

[3] ISO, DIN ISO/IEC 25000 Software-Engineering – Qualitätskriterien und Bewertung von Softwareprodukten (Software product Quality Requirements and Evaluation), 2010.

[4] B.P. Miller, L. Fredriksen, and B. So, "An Empirical Study of the Reliability of UNIX Utilities," *Communications of the ACM* , vol. 33, no. 12, December 1990.

[5] Wikipedia. (2011, August) http://de.wikipedia.org/wiki/Fuzzing

[6] Ari Takanen, Charles Miller, and Jared J. Demott, *Fuzzing for Software Security Testing and Quality Assurance*. Norwood: Artech House, 2008.

[7] ISTQB, "Standard glossary of terms used in Software Testing," 2007.

[8] L. van der Aalst, B. Broekman, M. V. T. Koomen, *TMap® Next - Ein praktischer Leitfaden für ergebnisorientiertes Softwaretesten*. Heidelberg: dpunkt, 2008.

[9] Wikipedia. (2011, Nov.) http://de.wikipedia.org/wiki/Scrum.

[10] (2011, Nov.) ISO/IEC 29119 Software Testing. http://softwaretestingstandard.org/