The Need for Functional Security Testing

C. Warren Axelrod, Ph.D., Delta Risk

Abstract. Despite extensive testing of application functionality and security, we see many instances of software, when attacked or during normal operation, performing adversely in ways that were not anticipated. In large part, this is due to software assurance staff not testing fully for "negative functionality," that is, ensuring that applications do not do what they are not supposed to. There are many reasons for this, including the relative enormity of the task, the pressure to implement quickly, and the lack of qualified testers. In this article, we will examine these issues and suggest ways in which we can achieve some measure of assurance that applications will not behave inappropriately under a broad range of conditions.

Introduction

Traditionally, software testing has focused on making sure systems satisfy requirements. Such functional requirements and specifications are expected to, but may not necessarily, accurately depict the functionality actually wanted by prospective users, particularly those aspects users may not be aware of or may not have been asked to consider.

In this article we examine the issues and challenges related to ensuring applications do not do what applications are not supposed to do. Such testing, for which we use the term Functional Security Testing (FST), is often complex, extensive and open-ended. And yet it is key to the secure and resilient operation of applications for the applications not to misbehave when subjected to various adverse stimuli or attacks.

The Evolution Of Testing

Programmers test applications that they are developing to ensure the applications run through to completion without generating errors. Programmers then usually engage in some rudimentary tests for correctness, such as ensuring that calculations correctly handle the types of data the programs process. In general, programmers seldom think "out of the box." This attribute was, to a large extent, the root cause of the Y2K "bug," where programmers frequently did not anticipate their programs would be running after Dec. 31, 1999 and so did not include century indicators in the programs, opting for two-digit year depictions. While it is true that programmers were motivated to abbreviate the year field by the need to stay within strict limitations in the amount of data that could be stored and transmitted, they failed to look to the future when their programs would crash.

During this early period, programs were thrown "over the transom" to the Quality Assurance or Software Assurance departments, where test engineers would attempt to match the functioning of the programs against the functional specifications developed by systems analysts. In general, such testers would limit their scope to ensuring the programs did what was intended, and not consider anomalous program behavior.[1]

Over the past decade, there has been greater focus on what might be called technical security testing, where security includes confidentiality, integrity, and availability. The usual approach is to assess the adherence of systems (including applications, system software and hardware, networks, etc.) to secure architecture, design, coding (i.e., programming), and operational standards. Often such testing includes checking for common vulnerabilities and programming errors, such as those specified by the Open Web Application Security Project (OWASP)[2] and SysAdmin, Audit, Network, Security (SANS) [3] respectively.

However, there are aspects of security testing that are different. For example, McGraw [1] refers to "anti-requirements," which "provide insight into how a malicious user, attacker … can abuse [a] system." McGraw differentiates anti-requirements from "security requirements" in that the security requirements "result in functionality that is built into a system to establish accepted behavior, [whereas] anti-requirements are established to determine what happens when this functionality goes away." McGraw goes on to say "anti-requirements are often tied up in the lack of or failure of a security function." Note that McGraw is referring to the adequacy or resiliency of security functions and not functions within applications.

Merkow and Lakshmikanth [2] refer to security-related and resiliency-related testing as "nonfunctional requirements (NFR) testing." NFR testing, which is used to determine the quality, security, and resiliency aspects of software, is based on the belief that nonfunctional requirements represent not what software is meant to do but how the software might do it. Merkow and Lakshmikanth [2] also stated that "gaining confidence that a system does not do what it's not supposed to do …" requires subjecting "… a system to brutal resilience testing."

In his book [3], Anderson affirms the importance of resilience testing with his comment that: "Failure recovery is often the most important aspect of security engineering, yet it is one of the most neglected." He goes on to note that "… secure distributed systems tend to have been discussed as a side issue by experts on communications protocols and operating systems …"

The author believes FST is another key area of testing that has received little attention from application development and information security communities and is not specifically mentioned in [1] or [2] or other publications.[4] Using FST, applications are tested to ensure they do not allow harmful functional responses, which might have been initiated by legitimate or fraudulent users, to take place. It should be noted here that testing for responses that do not derive specifically from functions within applications, such as when a computer process corrupts data, are not included in the author's definition of FST (see Introduction section).

It is important to differentiate among functional testing[5] of applications, which attempts to ensure that the functionality

of applications matches requirements; security testing, which aims to eliminate the aspects of systems that do not relate to application functionality but to the confidentiality, integrity, and availability of applications and the applications' infrastructure; and FST, which is designed to ferret out the malfunctioning of applications that might lead to security compromises.

In this article, we examine FST and how it relates to other forms of testing, look at why it might have received so little attention to date, and suggest what is needed to make it a more effective software assurance tool.

Categories Of Testing

Various types of testing are key for successful software development and operation, as discussed in [1], [2], and [3]. As described previously, software testers (or test engineers) most commonly check that computer programs operate in accordance with the design of an application and consequent functional specifications, which in turn are meant to reflect users' functional requirements. This form of testing is termed "functional *requirements* testing." When applications are tested for functionality in isolation, rather than in an operational context, the activity is called "unit testing." However, testers do need to ensure applications work correctly with the other applications with which they must interact. This latter form of functional requirements testing is known as "integration testing." And testers must further check that individual programs function appropriately in the various particular contexts to which they might be subjected. This further form of functional requirements testing is known as "regression testing," which is done to assure that changes in the functionality of an application do not have a negative impact on other components, subsystems and systems.

There is increasing interest from information security and risk management professionals in the security strength of software, as shown by the growth of such organizations as OWASP at <http://www.owasp.org>, which has seen very rapid growth in membership since its founding in 2001, and the Build Security In collaborative effort sponsored by the National Cyber Security Division of the U.S. Department of Homeland Security, at <https://buildsecurityin.us-cert.gov/bsi/home.html>. Such interest was largely precipitated by the discovery of a growing number of highly effective attacks against the application layer. Some estimate that such attacks account for as much as 70% of attacks.[6] As a result, we are seeing considerable growth in security testing of applications, as indicated by the increasing activity of organizations such as OWASP, mentioned above, as well as a strong movement for building security into applications from the beginning.[7] Security testing is essentially geared to reviewing software designs and coding practices, and the software code itself, to ensure the most egregious vulnerabilities have not been introduced into the concept, design, architecture, specifications, requirements, building, or release phases. When such exposures are discovered, they must be quickly eliminated. Security testing is essentially negative testing in that testers try to determine that certain attacks will not be successful. If the attacks are deemed to be a threat to a specific piece of software, then testers recommend they be eliminated.

While functional testing is commonplace, and security testing is increasingly popular, there is a third form of testing, which we term FST, which is not generally applied. This circumstance might be ignored were it not for the fact that this latter form of testing can be one of the most important aspects of software assurance. This is because it is this form of testing that provides some level of assurance that applications will not allow activities that should not be permitted. The desired assurance level depends on the cost and time (or any resulting delays) of the FST exercise and the benefits derived in terms of avoidance of malfunctions and resistance to attacks.

Some Examples Of FST

Perhaps the most ubiquitous example of the need for FST occurred during the Y2K remediation period. As the reader will recall, many computer programs, particularly within legacy systems, used two digit year data fields and did not recognize the date change to the 21st century. This might be considered a security issue as the integrity and availability of applications were at risk due to the Y2K problem, and the confidentiality of information was threatened in some situations. As a result, estimated expenditures of more than $300 billion were incurred to modify programs so they would correctly handle the millennium date rollover.[8] Despite these efforts, a large number of programs retained this defect and failed to process dates correctly, although catastrophic failure was avoided for the year 2000. For the most part, testing for the Y2K "bug" was a form of FST in that assessment of programs was on the basis of failure, not positive functionality, as previously mentioned.

More recently, there was a report of the interception of video transmissions from drones on surveillance missions in Iraq and Afghanistan where enemies could easily view video feeds using a $26 piece of commercially available software [9]. This is another example of systems that permit misuse because they were not tested against specific eventualities.

Increased Security Testing

About a dozen years ago, as the chief information security officer of a financial services firm, the author was asked to develop a series of scripts for testing the security of a strategic new client-server system. The testers had already created some 600 functional test scripts. The author came up with some 10,000 scripts for testing the "security" of the system. The number was large because there were so many possible combinations and permutations of ways to access functions and data. In the author's opinion, this ratio of testing scripts is typical.

During the intervening years, there has been increased interest in and growth of technical security testing, in the author's opinion, as a means of assuring the security of applications and systems. Such testing is critical to the secure operation of applications and does much to reduce vulnerabilities to attacks through the application layer. The range of such testing is very well described in several books, especially in [1]. However, this type of testing is performed by security testers who are familiar with the security weaknesses of programming languages and ways in which designers and programmers introduce vulnerabilities. These practitioners seldom get into the functionality of the applications and what security (i.e., confidentiality, integrity, and availability) exposures there might be in the functional operation of the software.

Skills Required Of Test Staff

The evolution of testing, from functional testing to security testing to FST, clearly relates to particular threats and the skill pools available at the time. In the early days of software development, the primary goal was to ensure the program worked and performed according to the requirements, with any residual errors showing up and being fixed during operations. Early mainframe and minicomputer systems were usually accessed and used by insiders or outsiders under the supervision of insiders. Programmers tested for the failure-free running of the software, whereas the testing staff ascertained that the functionality of the software was correct. Functional errors were turned back to the developers for correction and the software was retested until it operated correctly. In the author's experience, better-qualified test staffs were often familiar with the business use of the applications and had some rudimentary understanding of programming and computer operations.

With the arrival and proliferation of the Internet, applications were increasingly accessed and used by outside parties. This made for the need to test for a greater degree of security since organizations operating the systems often do not have much control over the actions of end users. The same deterrents that can be used against internal miscreants for their misuse of a system generally are not particularly effective against customers, business partners, or the public at large, since the latter are not subject to the same consequences, such as termination of employment.[9] Consequently, it has fallen to software vendors and internal staff to attempt to make sure that the systems cannot be compromised by evildoers both inside and external to the organization operating the software.

Table 1 shows the levels of knowledge, skill, and experience needed for each type of testing as well as how such attributes vary with the type of testing being conducted.

FST Issues

One of the issues relating to FST is that it does not yet have a generally accepted, immediately recognizable name. The type of testing referenced here lies somewhere between traditional functional testing and security testing, as we currently know these test categories. As described above, the former is used to establish that applications operate according to the functionality requirements expressed in the design of the system, that is, they do what they are supposed to do. The latter is a combination of tests relating to the security quality of the design, architecture, and coding aspects of an application, as well as other characteristics pertaining to the platform on which an application runs and the infrastructure that supports the system.

Other factors, such as the context in which the system will operate (e.g., Web facing, open, and internal), have a major impact on the level of testing that should be performed in each category. Criti-

Table 1: Required Knowledge, Skills, and Experience for Different Testing Approaches

Knowledge Skills and Experience Requirements for	Functional Requirements Testing*	Nonfunctional Requirements (Security) Testing**	Anti-Requirements Testing***	FST****
General business	Moderate	Low	Low	Moderate
Business processing	Moderate – High	Moderate – Low	Low	High
Coding standards	Moderate – Functional coding standards	High – Security coding standards	High – Security coding standards	High – Functional and security coding standards
Testing Methods	High – Functional testing	High – Security testing	High – Security testing	High – Functional and security testing
Computer operations	Moderate	High	High	High
Security – Privacy and Confidentiality	Moderate - High	High	High	High
Security – Integrity	Low to Moderate	High	High	High
Security – Availability	Low	High	High	High

* *Per McGraw [1], Merkow and Lakshmikanth [2] and Anderson [3]*
** *Per Merkow and Lakshmikanth [2]*
*** *Per McGraw [1]*
**** *Per this article*

cal Web-facing commercial applications and embedded systems operating aircraft or weaponry must undergo extensive testing.

Between these two traditional test modes, are more recent approaches. One is called "anti-requirements testing" or "negative requirements testing"[10] of applications. This is directed at the security aspects of systems and its purpose is to prevent security-related components from behaving badly.

The fourth category is FST, which is essentially testing to ensure the application does not allow application functionality and data use that is forbidden, either implicitly or explicitly, which might compromise security. An example of such a test objective would be … "Do not allow one customer to see another customer's data." To test this fully, every possible combination of user access to functions and data, both authorized and unauthorized, must be tested, which is impossible in practice. Therefore, some compromises must be made as to how much of this testing can reasonably be done, subject to project time and funding constraints.

Why So Little Attention To FST?

The author believes that there are a number of reasons why there is insufficient emphasis on FST, namely:

- FST can be orders of magnitude greater in effort and cost than traditional application functional testing and security testing.
- To perform FST properly, testers must be knowledgeable and experienced in business functions and application security, as well as software assurance processes.[11]
- The importance of testing for negative functionality is, in the view of the author, not generally recognized by general business management, risk managers, IT management, and application development managers.

What Needs To Be Done?

Support from Owners and Participants:

The most important step is to gain support for FST and get various participants and owners to understand that there is a very significant gap in standard software testing. This gap shows up when, for example, insiders are able to access information to which they should not have access, and then use the information for nefarious purposes.

FST Scope and Procedure:

The scope, procedure, time frame, and cost for a particular FST exercise have to be determined in advance. In many cases, the cost of running through all possible FST scripts is prohibitive and cannot be justified economically. Therefore it is necessary to create an iterative, adaptive procedure. One such approach is to test random samples from the entirety of test scripts and determine, using statistical methods applied to the design of experiments, from the results of the sample tests whether it is worthwhile to test further samples. In the author's experience, this approach results in running a relatively small subset of the complete list of test scripts, which usually reduces the cost and duration of testing considerably, while improving the accuracy of the results.

The author recommends an FST methodology which consists of the following steps:

1. Review test scripts that have been created for functional requirements testing.
2. Create as complete a set as possible of potential user activities (use cases) and remove those activities that are included in the functional requirements test scripts.
3. Develop test scripts for the remaining activities.
4. If the number and size of test scripts of the remaining activities are too many and too large, respectively, to justify the expense and time for such testing as determined by a return on investment (ROI) analysis, adopt a method for testing a succession of random samples of the scripts (where the size of the sample is based on ROI) and run the selected scripts.
5. If the error rates are significant then the sample size should be expanded based on the risk suggested by the prior tests.
6. The application and system errors, which are detected in the FST process, are fed back to the development team for correction and the corrected code is then retested until all material errors are fixed.

ROI:

Regarding ROI, the cost of and time required for performing tests can generally be estimated with a reasonable degree of accuracy. However, with FST and NFR (security) testing, the dependency of future sample sizes on the results of prior tests makes for dynamic costs and durations. In addition, the benefits of FST and NFR testing are particularly difficult to determine since, when sampling is involved, it is not clear what errors might remain.

In practical terms, FST should continue until the development and testing teams are reasonably satisfied that the applications no longer harbor any major latent deficiencies, subject to maintaining a positive ROI from the FST process itself.

Body of Knowledge:

Finally, we need to develop a body of knowledge about FST from experience with successful tests and actual software failures. As the library of such tests expands, it is important to share test results with others who can then apply lessons learned to their own FST programs. In this way, testers can more readily determine which types of tests are likely to be more fruitful and these testers, in their turn, can contribute new FST facts and experiences to those already cataloged.

Summary and Conclusions

The testing of applications to try to ensure that they do not misbehave is complex, sophisticated, and expensive. Yet the cost of not doing such tests, in terms of security incidents, can be so much greater than going through a realistic FST exercise.

There may well be some centers where this type of testing is already being performed, but is not called FST. However, the author believes it is safe to say that FST is not ubiquitous, as can be seen from the flood of incident information that often appears in the news.[12]

It is also apparent from the lack of published material in this area that developers and security professionals are not generally familiar with the principles of FST and therefore do not practice them to the detriment of system confidentiality, integrity, and availability. The situation will only improve when it is generally accepted that we need to ensure applications are prevented from functionally allowing damaging events. This is in addition to the non-functional security testing that is more commonplace.

ABOUT THE AUTHOR

C. Warren Axelrod, Ph.D., is a senior consultant with Delta Risk, a consultancy specializing in cyber defense, resiliency and risk management. Previously, he was the chief privacy officer and business information security officer for US Trust, the private wealth management division of Bank of America. He was a co-founder of the Financial Services Information Sharing and Analysis Center. Dr. Axelrod won the 2009 Michael Cangemi Best Book/Best Article Award for his article "Accounting for Value and Uncertainty in Security Metrics," published in the Information Systems Audit and Control Association (ISACA) Journal, Volume 6, 2008. He was honored with the prestigious Information Security Executive Luminary Leadership Award in 2007. He received a Computerworld Premier 100 IT Leaders Award in 2003.

Dr. Axelrod has written three books, two on computer management, and numerous articles on information technology and information security topics. His third book is *Outsourcing Information Security*, published in 2004 by Artech House. His article "Investing in Software Resiliency" appeared in the September/October 2009 issue of CrossTalk magazine.

He holds a Ph.D. in managerial economics from Cornell University, as well as an honors M.A. in economics and statistics and a first-class honors B.Sc. in electrical engineering, both from the University of Glasgow. He is certified as a Certified Information Systems Security Professional and Certified Information Security Manager.

C. Warren Axelrod, Ph.D.
Delta Risk
P.O. Box 234030
Great Neck, NY 11023
E-mail: waxelrod@delta-risk.net

REFERENCES

1. McGraw, Gary, Software Security: Building Security In, Upper Saddle River, NJ: Addison-Wesley, 2006, pages 213-216.
2. Merkow, Mark and R. Lakshmikanth, Secure and Resilient Software Development, Auerbach Publications, forthcoming September 2010.
3. Anderson, Ross, Security Engineering, Second Edition, Indianapolis, IN: Wiley Publishing, 2008, pages 192 and 212.
4. Hass, Anne Mette Jonassen, Guide to Advanced Software Testing, Boston, MA: Artech House, 2008.
5. Michael, C.C. and Will Radosevich, "Risk-Based and Functional Security Testing," Cigital, Inc., 2005, at <https://buildsecurityin.us-cert.gov/bsi/articles/best-practices/testing/255-BSI.html>
6. Gegick, Michael, "Intro to Security Testing," NC State University, 2006 at <http://resist.isti.cnr.it/free_slides/security/williams/Security1.pdf>
7. "Application-layer security: What it takes to enable the next generation of security services," Nortel White Paper at <http://www.nortel.com/products/01/alteon/2224/collateral/nn105560-010605.pdf>
8. "Application-layer Attack Protection: Proactive defenses for your critical business applications," MacAfee Web Page at <http://www.macafee.com/us/enterprise/solutions/network_protection/application_layer_attack_protections.html>
9. Shane, Scott and Christopher Drew, "Officials Say Iraq Fighters Intercepted Drone Video," The New York Times, December 17, 2009 at <http://www.nytimes.com/2009/12/18/world/middleeast/18drones.html>

NOTES

1. I recall an exceptionally talented QA manager, who reported to me in the 1980s and who would spend a day or so after the functionality of the system had been assured just hitting keys at random to see how the system would respond. He invariably found program errors that had not shown up in the original functional testing. Today this might be called "fuzz testing."
2. See OWASP Top 10 – 2010 (Release Candidate 1) at <http://www.owasp.org/images/0/0f/OWASP_T10_-_2010_rc1.pdf>
3. See CWE/SANS Top 25 Most Dangerous Programming Errors at <http://www.sans.org/top25-programming-errors/>
4. There are occasional references to functional security testing such as in Hass's Guide to Advanced Software Testing [4], page 248, Michael and Radosevich's article "Risk-Based and Functional Security Testing" [5], and a presentation by Michael Gegick [6]. In some cases, there is confusion between functional security testing and what the author would term "security functionality testing," which is done to ensure that the security functionality, rather than the application functionality, is correct. Even when functional security testing is defined the same way as it is in this article, as in [4], little detail is provided as to the scope of the testing effort and the use of sampling to make the testing manageable.
5. In the commercial world, "functional testing" is sometimes referred to as "business logic testing."
6. The percentage of attacks affecting the application layer (as opposed to networks) is variously estimated in the 70 to 80 percent range, as in the Nortel White Paper [7] and the MacAfee Website [8]. The accuracy of these numbers is highly questionable since, in the author's opinion, the vast majority of compromises of applications are never detected, especially those promulgated by insiders, such as employees, contractors, and service providers' staff. However other forms of attack are also underestimated. Perhaps the best one can say is that security professionals believe that a large percentage, possibly the majority, of compromises are those related to applications.
7. Descriptions of the various BSI (Build Security In) approaches can be found at <https://buildsecurityin.us-cert.gov/bsi/home.html>, <http://www.bsi-mm.com>, <http://www.opensamm.org>
8. See "Y2K: Overhyped and oversold?" BBC News, January 6, 2000, at <http://news.bbc.co.uk/2/hi/talking_point/586938.stm>
9. In a comment on the author's Bloginfosec column "Insider Threat – Not Knowing That You Don't Know What You Don't Know," available at <http://www.bloginfosec.com/2010/05/10/insider-threat-%e2%80%93-not-knowing-that-you-don%e2%80%99t-know-what-you-don%e2%80%99t-know/2/>, Gary Hinson raises the issue of "plausible deniability." He contends that the "wayward insider" is better able to claim that the unauthorized activity was an accident. This reduces the deterrent value of disciplinary consequences against the employee or other insider.
10. Dr. Herbert (Hugh) Thompson, Chief Security Strategist at People Security, coined this term.
11. Such skills can be demonstrated by testers having appropriate certifications.
12. The author recently spoke with an experienced software engineer, who is involved in the design and development of safety-critical systems. He described the way in which he tests such systems. His approach was very similar to that defined as FST in this article. However, he had not formalized such testing as markedly different from regular functional testing, and he was not aware that this approach was not widely used. This suggests that there are pockets of testers performing FST, but such centers of excellence are not common as demonstrated by the frequency of software failures.