

Preamble

This material is released under a Creative Commons License agreement <http://creativecommons.org/licenses/by/2.0/uk/>. In informal terms we, the authors, give you permission to do most things with the material provided you:

- a) Acknowledge us as the authors of the original material, and retain the acknowledgement in copy of the material, and in any of your material based on ours,
- b) Do not pretend you wrote it,
- c) Do identify any changes you make as your work, not ours.

The material includes opinions of the authors, and may be different from work published by others. We have quoted and referenced work from third parties, and in a few cases have knowingly used or adapted ideas or content from third parties. Where appropriate we ask for permission from the third parties.

However we make mistakes, as most people do, and may have made mistakes in this work. If you spot something you feel is wrong, or simply something you feel could be improved please let us know via this email address nft.introduction@commercetest.com

We are Stuart Reid and Julian Harty, and are the authors of this material: **An introductory course on non-functional software testing**. If you wish to create derived works you need include the following acknowledgement: at the start and end of the derived works: **This work is based on original material by Stuart Reid and Julian Harty**. We like to know whether our work is being used so please tell us how you're using the content.

Finally, this work is still in a draft form so you may find inconsistencies, gaps, incomplete content, etc. We hope you will find the content useful anyway and we are very happy to receive suggestions for improvement to the email address mentioned above.

Software Security

Security testing is a hot topic at the moment. With the prevalence of commerce on the Internet and web-based applications security has emerged as one of the most important issues that need to be addressed by the software industry. Security testing needs to be adopted as a mainstream activity. Testers need to have a mindset that allows them to attack software to expose vulnerabilities.

Here is a brief definition of security testing:

Security testing definition

- *Testing whether the system meets its specified security objectives*
BS7925-1:1998

So what are the security objectives?

The security objectives includes three vital aspects of a computer system:

- Confidentiality
- Integrity
- Availability

You can use the simple acronym of “CIA” to remember these three aspects.

There are various security mechanisms that are used to implement these areas, such as:

- Authentication
- Authorisation
- Auditing

All these mechanisms begin with the letter A (in English at least) and are commonly referred to as 3A's in literature on security.

Security testing has great deal of overlap with **robustness testing**.

Robustness testing determines whether the software is able to continue to meet other requirements in undesirable or adverse conditions. Examples of the types of things robustness testing includes are:

- Invalid inputs
- Failure of a supporting component (either hardware e.g. a disk drive, or software e.g. a library of software code)

If software is not robust, failures may occur in undesirable or adverse conditions, these failures can affect the integrity of the system, availability for users, etc. The effects may be limited to one user, or many users.

Robust software needs mechanisms to:

- Prevent
- Detect
- Isolate
- Correct
- Recover from

undesirable or adverse conditions.

Secure software needs similar mechanisms to ensure:

- The system maintains the confidentiality and integrity of data and operations
- The system is available for authorised activities from authorised users, and not available for unauthorised users or unauthorised activities.

There is a sometimes-subtle distinction between the things we look for when doing security testing compared to robustness testing:

- Security testing tries to find failures in:
 - Confidentiality e.g. information leakage
 - Integrity e.g. using spoofing to fool software to accept input it should reject, or by corrupting data held in the system
 - Availability, e.g. by denying access to authorised users

Security testing may use robustness testing techniques to find the failures, and then 'looks-at' the effect of the failure.

- Robustness testing tries to find failures caused by:
 - Invalid inputs
 - Undesirable conditions or events

And is less concerned about the effect(s) of the failure. However the recovery process is of greater interest than it otherwise might be for security testing.

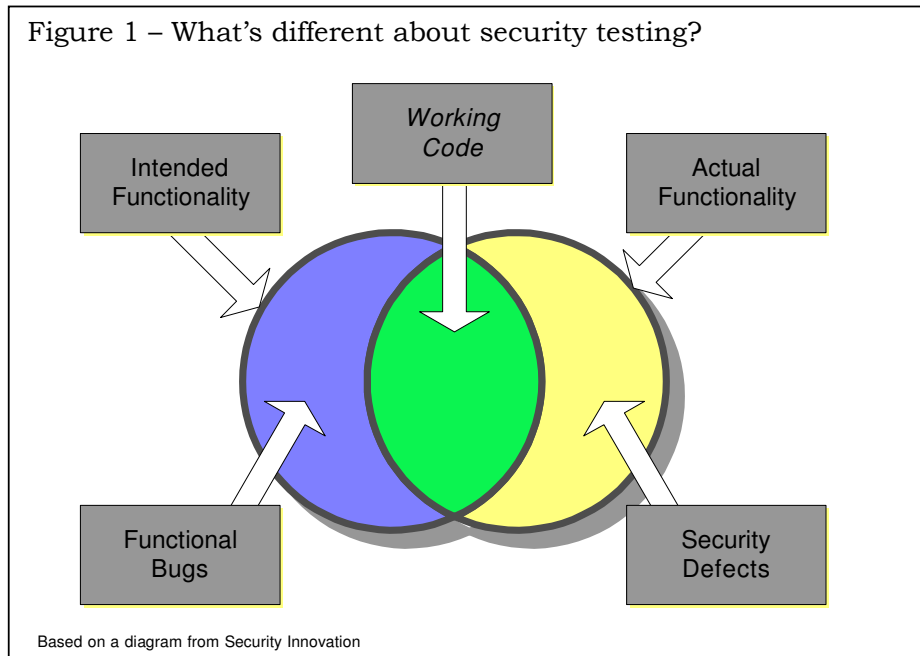
Security testing shares many common test techniques with robustness testing, and the testing needs to also include tests to determine whether the confidentiality requirements are also being met.

When dependability is compared with security there are a number of common terms e.g. Availability. Dependability testing is covered in the software dependability section of this material. Differences between the two are covered later in this section.

Why is security testing different?

From functional testing?

Figure S1 shows that security testing differs from functional testing as it concentrates of finding things that *shouldn't occur, but do!*



Security testing requires a different mindset from traditional functional testing, as the tester has to consider ways to test for these things that shouldn't work e.g. using negative testing techniques, threat-modelling, etc.

Comparing security testing with...

Performance testing

Stress testing, which is often considered part of performance testing, may overload a system and as a result the system may fail one or more of its security requirements e.g. by disclosing confidential information, or denying service to authorised users. Therefore stress testing may help expose security bugs.

Usability testing

Users are more willing to use a system they believe is secure, so a secure system may be more usable. Therefore the results of security testing may have a positive impact on the perceived usability.

However increased security may make the system 'harder' to use e.g. by forcing users to revalidate their security credential from time-to-time while using the system, which may have a negative impact on perceived usability.

The perceived usability may then be recorded in usability testing techniques such as SUMI and WAMMI, which are introduced in the usability section of this material.

Dependability testing

Both dependability testing and security testing measure the trust placed by users and stakeholders in a computer system. However the two types of testing use different techniques to measure different qualities of the system. For example:

- Availability in the context of security includes the constraint ‘for authorised users’, and the inverse: that the system should not be available for unauthorised users.
- While availability in the context of dependability is ‘is the system there when I need it?’ regardless of whether ‘I’ represents an authorised or an unauthorised user.

Special stakeholders

Company auditors, both internal and external, have a special interest in whether computer systems are secure e.g. they need to know whether critical or sensitive data can be modified, leaked or stolen.

Many countries have legislation that implies software needs to be reliable and secure e.g. the Data Protection Act in the UK (1998); the Freedom of Information Act in the UK (2000); Sarbanes-Oxley (SABOX) in the USA, etc. Such legislation tends to hold company directors responsible for problems, therefore company directors have an interest in ensuring systems are secure, and therefore that the security has been tested.

Some security testing requires the testers to be extremely technically competent e.g. for black-box reverse engineering of systems using decompilation and disassembly techniques.

Benefits and Risks associated with security testing

Security testing allows people to gain some confidence in whether the software can be:

- Available when required,
- Protect sensitive data,
- Trusted with the results it provides.

Adequate security testing helps to provide a degree of confidence for: auditors, directors, and others that they have met their statutory requirements.

However security testing also brings attendant risks, e.g.

- Can we trust the people who did the penetration testing to tell us everything and tell no one else anything?
- As we are trying to defend against shady, and sometimes nameless characters, how do we know whether we've found all the important security issues?
- Do the testers have the right skills and mindset to look for security issues? Are they competent in using security testing techniques?
- Do the software tools find all the faults they claim to?
- How will we know when we have done enough, a question that's particularly pertinent when security testing can be expensive and time-consuming!

Costs of getting security wrong

Security breaches can be very, very expensive, embarrassing, and even threaten the future of the business. Directors may face criminal charges, and staff be disciplined in the event of a serious security vulnerability being exposed in public.

Examples of security not being specified / tested adequately

There are numerous examples of security issues causing outages, information leakages, etc. Here are a few examples of security issues that became public; you can find many more by searching the Internet.

SQL Slammer

In January 2003, SQL Slammer spread rapidly around the Internet, and within a few hours had swamped parts of the network, and taken 1,000's of servers on the Internet offline by attacking those running Microsoft's SQL Server 2000.

The cost to fix the problem alone was estimated at over \$750m; the costs related to the loss of productivity were far higher. Many companies were embarrassed as their critical web servers stopped working.

“LoveLetter”

Love Letter was a basic virus that spread by email using a Visual Basic script to propagate copies of itself to other email addresses found in each recipient's address book. Apart from the estimated \$1B cleanup costs, and

\$7.7B lost productivity, 1,000's of people ended up buying anti-virus software, something they hadn't considered necessary before.

UK utility company

A major utility company in the UK stored details of over 7,000 customers on their web site. The data included addresses, credit card and payment details. Although the company initially threatened to sue the person who discovered the data and who reported the problem to the press, the company was forced to apologise to their customers.

Potential conflicts

With functionality

Security should not conflict with functionality, as the goals of security are consistent with the goals of stakeholders e.g. the software should work correctly for authorised users, and deny service for unauthorised users. However designing, implementing and testing security tends to be expensive and time-consuming for companies currently, and a side effect of having to address security requirements may reduce the functionality that ends up being implemented.

Also, there is a trade-off between providing lots of functionality when software is installed, out-of-the-box, and having the software install with all non-essential functionality disabled (to reduce the chances of the software being breached through functionality the user may not want, or even be aware of).

With other attributes

If security is considered in isolation, or if security is not considered at the same time as other non-functional attributes are being considered, security can easily conflict with other non-functional attributes.

Examples of specifying security requirements

Here is an example of how a security requirement for an ATM may be specified to meet the SMART specification for non-functional requirements.

ID	NFR.17
Description	Storage of PIN in ATM memory The PIN is sensitive information and could be combined with the ATM card, or a facsimile, to perform unauthorised transactions (e.g. to steal money from someone's account). Therefore the ATM must not retain the PIN once the user completes their transaction(s).
User Requirement	UR.1
Primary Attribute	Security.
Measure	Pass/Fail.
Basic Requirement	No trace of the PIN is found in the ATM, including but not limited to the internal memory, once the user has completed their activities¹.
Ideal Requirement	As above.
Current value	Not applicable.
Technique	Static code analysis tools, Memory analysis tools
Test Requirement	ATM set up with an in-circuit emulator and memory analysis tools. A bank server simulator.
Operational Requirement	Monitor any bug or security reports.
Conflicts	None.
Priority	6 (on range 1-10 for this project).

Note: Security requirements may blur the distinction between requirements and implementation, e.g. when a requirement mandates the use of SSL for sensitive requests to a web site. From a purist's perspective this is undesirable, however practitioners often face these sorts of requirements and learn to work with them.

¹ Note: A user may complete their transaction in a number of ways, e.g. by pressing the 'return card' menu option, by selecting 'no' to 'do you want another service'. Please see the full functional specification for the comprehensive list.

Approaches to testing security

When is security testing performed in the SDLC?

Security testing is relevant throughout the software development lifecycle. Some techniques are relevant at specific stages in the lifecycle; others can be used throughout. The following paragraphs introduce some common security testing techniques.

Threat modelling is a technique that can be applied throughout the SDLC and should be started as soon as practical. Threat modelling helps testers, and others, to “think like an attacker”, to discover threats, and to help manage those threats.

Code reviews are used both during and after software is written to help find and fix potential security vulnerabilities. Code reviews review the source code, and may include:

- Automated tools. The tools are driven by a set of rules that specify areas of known weaknesses. The output has to be interpreted by competent programmers to identify which of the possible problems are worth further investigation.
- Competent programmers reviewing the code directly to look for new, or recognised flaws for the programming language and technologies being used for this software.

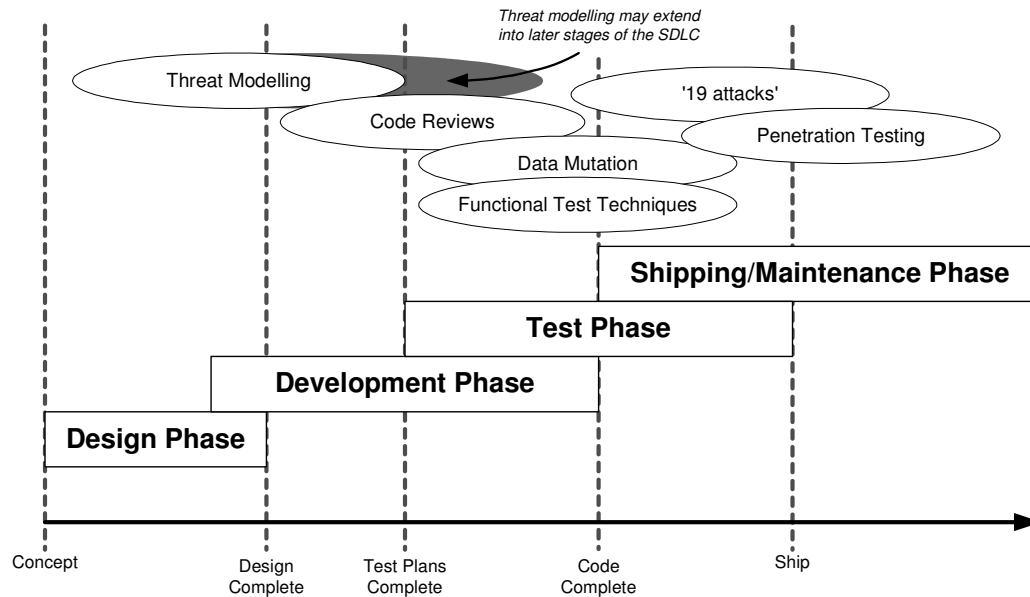
There are a number of test case design techniques, including:

- **Data mutation,**
- Adapting common ‘functional’ test techniques,
- And a set of **19 attacks** from Security Innovation, further information is available from the references.

Security testers can use these test case design techniques during the various test phases of the project e.g. during system testing.

Attack techniques, e.g. penetration testing are performed against a running copy of the software to see if the software defences can be compromised.

The following figure illustrates where various security testing techniques fit in the SDLC.



Security testing techniques through the Software Development Life Cycle (SDLC)
Based on ideas from Microsoft's Security Development Process

Where is security testing performed?

Security testing can be performed wherever software is! It can even be performed against production systems (after all this is where the majority of systems are attacked in real-life). However responsible testers will limit any such testing of production systems to the minimum required and they will design their tests so the tests do not damage the underlying systems or data.

Note: production systems tend to be configured differently from test systems: such configuration changes could introduce security flaws. Therefore some sort of testing of and/or monitoring is important for finding security issues in production.

When more intrusive or destructive testing is required, the testing should be performed on an isolated system in an isolated test environment. Critical data, etc. should be backed-up before the testing starts, the backups help to quickly and reliably rebuild compromised, or broken, test environments.

When learning security testing, testers will appreciate having a safe, isolated test environment. Safe environments include things like an isolated computer (with no network connection), or **virtual machines** can provide a similar facility if they are configured appropriately.

What special resources are required?

Simple tools are sufficient for the majority of security testing e.g. a text editor that includes facilities to edit source code, a web browser, and most importantly an intelligent, competent security tester ☺

More specialist tools, such as network analysers, disassembly tools, and security testing tools, will help particularly for analysis and penetration testing.

Costs of security testing

Commercial, outsourced security testing tends to be expensive, and the results hard to rate in terms of value for money. However, internal security testing may be time-consuming, and equally unreliable. “In my opinion a lack of understanding of the subject of security testing is responsible for both of these situations.” [Julian Harty].

By increasing our understanding of security testing, and software security in general costs can be reduced and the effectiveness of the money spent increases.

References

Here are some references related to security and security testing.

References from books and other printed material

- S-P 1. Writing Secure Code, second edition; Michael Howard and David LeBlanc; 2003 – Chapter 19 on security testing is particularly useful, so is the rest of the book!
- S-P 2. Software Testing Techniques, Second Edition; Boris Beizer; 1990 – Still available second-hand and one of the standard references on software testing.
- S-P 3. How to Break Software Security; James Whittaker Herbert H. Thompson, 2004 – a great, slim book with practical techniques for attacking software security.

References from the Internet

- S-I 1. <http://www.securityinnovation.com/chart.htm> - Why security bugs are different.
- S-I 2. <http://www.cigital.com/ssw/presentations/howard.ppt> - Michael Howard's presentation where he mentions 50% of security bugs are found using threat analysis.
- S-I 3. http://en.wikipedia.org/wiki/Penetration_testing - explains penetration testing.
- S-I 4. <ftp://ftp.info.ucl.ac.be/pub/publi/2003/avl-RHAS03.pdf> - Scientific paper on Anti-goals.
- S-I 5. <http://easyweb.easynet.co.uk/~iany/consultancy/papers.htm> – Material on misuse cases, etc.
- S-I 6. Writing Secure Code, second edition; Michael Howard and David LeBlanc; 2003 – Chapter 19 on security testing is particularly useful, so is the rest of the book!
- S-I 7. Software Testing Techniques, Second Edition; Boris Beizer; 1990 – Still available second-hand and one of the standard references on software testing.
- S-I 8. http://www.testingstandards.co.uk/bs_7925-2.htm - link to a document containing the common functional testing techniques.
- S-I 9. <http://www.securityinnovation.com/resources/attacks/index.shtml> - Introduction to the 19 attacks.
- S-I 10. <http://www.owasp.org> - great web site with free applications for web-based security testing, experimentation, and useful documents.
- S-I 11. http://news.com.com/Counting+the+cost+of+Slammer/2100-1001_3-982955.html - the cost of SQL Slammer and Code Red
- S-I 12. <http://www.cigital.com/papers/download/bsi8-program.pdf> - Security testing techniques through the SDLC
- S-I 13. <http://www.cert.org/advisories/CA-2000-04.html> - I Love You email worm
- S-I 14. <http://www.net-security.org/press.php?id=1635> - most expensive viruses to 2003
- S-I 15. <http://www.silicon.com/software/security/0,39024655,11018650,00.htm> - Powergen security breach exposes customer details.
- S-I 16. http://downloads.securityfocus.com/library/beagle_lessons_2.pdf - very useful background reading on the power of a professionally developed virus, called Beagle. And part 3 of the story is here (from the original author's web site) http://www.infectionvectors.com/library/year_of_the_beagle.pdf

-- End of document --