Security Testing: Turning Practice into Theory

Sven Türpe
Fraunhofer Institute for Secure Information Technology SIT
sven.tuerpe@sit.fraunhofer.de

Abstract

This position paper proposes a research agenda for the field of security testing. It gives a critical account of the state of the art as seen by a practitioner and identifies questions that research failed to answer so far, or failed to answer in such a way that it would have had an impact in the real world. Three categories of research problems are proposed: theory of vulnerabilities, theory of security testing, and tools and techniques.

1. About this Paper

The science of security testing is still in its infancy. This paper proposes a research agenda for this field. It does so from a very specific perspective: that of a tester who, being aware of the lack of a scientific basis of his work, has to and wants to assess the security level of software systems on the basis of testing. What such a tester needs is not research papers but useful tools that optimize the work that is already being done in various labs around the world. The key underlying assumption of this paper is therefore that research should take an approach similar to what a usability engineer would do when designing a tool: first understand the task, then design solutions and tools. Hence the title, *turning practice into theory*.

This paper contains no original research whatsoever. Rather, it is a position paper and conveys the author's opinion on the subject. The author has a background in applied research and practical security testing, which may explain some of the views expressed here. Primarily, the present paper collects problems the author encountered during several years of testing and evaluating systems for their security. Secondarily it presents a number of observations how security testing is approached today, none of which should be taken for more than anecdotal evidence, though.

The remainder of this paper is organized as follows: Section 2 outlines the author's conception of security testing. This section serves as an extended introduction as the readers' backgrounds and views on the topic may vary. Sections 3 through 5 propose research problems n three different areas. Section 3 is dedicated to vulnerabilities as such, section 4 deals with the conceptual side of testing, and section 5 is about actual tools that might be useful in the field. Finally, section 6 wraps up the paper and draws a short conclusion.

Since this is a position paper the author does not attempt to provide references for the individual views expressed or dismissed. References are provided where they are available and helpful for understanding or recommended for further reading.

If not specified otherwise, the considerations in this paper primarily apply to the security testing of application software. This does not imply any particular assumptions but should be considered if claims made here seems to contradict results that had been achieved for special cases. Claims and suggestions made here may be wrong, in particular, for very small systems, such as a smart-card operating system; for very restricted concepts of security or vulnerability, e.g. testing for buffer overflows and nothing else; and for security systems, i.e. systems whose sole purpose is to enforce security policies, such as firewalls or security mechanisms of operating systems.

2. Real-world Security Testing

There may be different views what security testing is or is not. This section outlines the author's view without claiming that it would be the only, the most appropriate or the most complete one, or in any other way special. The sole purpose of this section is to provide some mental background for the statements to follow.

2.1. What is Security, Anyway?

Security is one of the many aspects of software quality. A piece of software could be functionally cor-

rect yet lack quality: usability, stability, security, or others. As a preliminary definition, we may characterize security as follows:

Software security is the absence of properties and features that pose a risk to the operator of the software or third parties if they are exploited with malicious intent.

Note that properties and features at this point may include a wide range of things from the way a piece of software performs access control to the way it interacts with the user. If something can be exploited, it is covered by this definition. Excluded, on the other hand, is anything that does damage through mere accident.

This definition has a number of implications. First of all, this definition puts emphasis on exploits rather than security functions. Counting security functions obviously does not determine the quality of their implementation and thus, their resistance against malicious attacks. It is also a well-known fact that real-world security issues rarely appear within the security functions of a system. Buffer overflow errors, for instance, may appear anywhere in a system and their impact usually does not depend upon security functions at all.

Second, the definition is designed to directly conflict with the concept of testing. Testing, as we are all aware, can never show the absence but only the presence of errors. Yet security ultimately is about the absence of certain types of errors. Whether this is an issue or just a matter of language shall remain open here.

Third, this definition – wrongly – follows the all too common practice of ignoring the attacker's motivations and all economic considerations. It does so very rigorously, though, in that it also ignores what is commonly known as security objectives. Well, not entirely: such considerations are buried under the seemingly innocent terms risk, exploit, and malicious. More in-depth considerations shall remain in the background for the moment as common notions such as confidentiality – integrity – availability may do little to guide efficient security testing.

All in all, the definition emphasizes what makes security testing hard: the fact that one is supposed to test for something very unspecific. Security testing is about finding things not specified, before somebody else finds them.

2.2. Motivations and Constraints

Real-world security testing has real-world purposes. Its point is not, usually, to find some obscure instance of some obscure class of bugs for which one happens to know a testing technique. Who does security testing, why are they doing it and what matters to them? What are (some of) the constraints under which security testers commonly have to work?

Security testing may occur in such diverse positions as: development or quality assurance on the part of a manufacturer; operators of a system; various 3rd parties, such as administrative bodies, the media, research, or expert witnesses in a court hearing; as an adversary testing with malicious intent; or in an independent laboratory on behalf of any of these roles. One important implication of the tester's position is side conditions. While a developer for instance naturally has access to source code a 3rd party or operator may not, yet face the same questions about security.

Consequently, there is a wide range of possible objectives that the tester might pursue: find one issue that can be exploited or find all of them; find issues anywhere or in a specific part, subsystem or function; find defects of a specific type or anything that could be exploited; give a rough assessment or provide sufficient evidence for a security certificate; test an individual target or compare several of a similar kind; make the general public believe that one product is more secure than another.

These enumerations are likely incomplete but sufficient to suggest that one single approach might not serve all needs, and that a real-world tester will hardly ever encounter ideal conditions for security testing that are assumed by so many research papers. To be useful in practice, a testing technique or tool must work at least for one combination of position, side conditions and objective.

Depending on their situation, security testers may be subject to a number of constraints. Besides the common resource limits – CPU power, memory, time, money, network bandwidth, etc. – they are often facing a particular issue. Security testers often have to work with incomplete information about a system, thus being in a similar situation as the adversary.

2.3. Security Testing Today

As far as practice is concerned, a science of security testing seems non-existent. The only notable exception may be security certification which, however, tends to be focused on security functions and ignores most of the interesting (read: difficult) issues. A system may be security certified yet insecure, as for instance the case of the Xerox WorkCentre printer has shown. This device passed a Common Criteria evaluation [10] but was demonstrated to have severe vulnerabilities just a few months later in a BlackHat conference presentation [17].

In the field, people do what they can, and this is not much. Ad-hockery and makeshift tools seem common.

Common techniques generally known or observed by or reported to the author include:

- Checklists of varying depth and quality. They may contain items as unspecific as: all input should be sanitized.
- Generic tools such as monitoring tools (e.g. for network traffic, file access etc.), interfacing tools (e.g. Netcat [5], Socat [9], Scapy [8], etc.) and programming languages (often scripting languages such as Perl, Ruby, Python, and others).
- Fuzzing [16], sending more or less random input to an interface in the hope of hitting a bug somewhere by chance. Alas, fuzzing is about how to look but not what to look for. A more targeted way of fuzzing is also called fault injection [22].
- Vulnerability scanners, particularly for Web applications, whose performance is generally rather poor. They tend to miss important issues and to produce too many false positives [18,23].
- Re-use of functional test cases and their modification into security test cases, e.g. by changing test inputs in such a way that they might trigger further error conditions.
- People running Nessus (yes, *that* Nessus [4]) or other inappropriate tools on every system purchased.
- Hacking or hiring hackers, hoping they would know more than we do. They don't usually, but hackers produced the majority of the testing tools that we know today.

If science has had any proposals for better tools, they haven't found their way into the security labs and development shops out here. This point is supported by a look into recent textbooks. In the 2006 edition of [15] for instance just 15 out of about 400 pages are dedicated to software penetration testing, and [24] mentions only 3 recent papers on security testing.

We know on the other hand that hackers of whichever hat color are successful in security testing at least to the extent necessary to find some vulnerabilities sometimes, so there must be something that can be done and is being done.

2.4. Requirements for Useful Tools and Techniques

It would be a research project on its own to gather and specify detailed requirements for testing techniques, tools and frameworks that will work in the real world. A few suggestions are appropriate for this paper, though.

First of all, useful tools must address the right problem(s): help us to test for the issues that commonly appear in software and are not easily fixed by using a more programmer-friendly platform. If you think of the OWASP Top Ten [6] at this point, you are probably right.

Second, they must not require idealized side conditions. Techniques are needed that continue to work under the adverse conditions often encountered in security testing. They must be robust and work well with incomplete information about the target of testing.

Third, tools and techniques must provide guidance to their users throughout the testing task. While generic interfacing tools are the most useful today, they don't do anything to help their users design test cases.

Fourth, results must be useful under real-world conditions. The vulnerability scanners that we have today don't live up to this standard.

Fifth, tools and techniques should provide, or be based on, useful abstractions. In particular they should provide abstractions from implementation detail, such as the particular languages or network protocols involved, where the issues tested for are structurally similar.

Real-world testers might be willing to sacrifice rigor and formality to robustness, usefulness and usability.

3. Understanding Vulnerabilities

The basis of all systematic security testing must be a comprehensive theory of vulnerabilities. This is something we miss almost entirely today or if it does exist, it hasn't reached the practitioners in the field. We do not even share a common terminology, we rather invent new terms whenever there is an opportunity to do so. The world of security testing today is full of special cases with melodious names and phenomenology but largely free of meaningful concepts and abstractions. To develop more appropriate ways of talking about vulnerabilities is therefore one of the research topics proposed here.

3.1. Descriptions, Classifications and Abstractions

Meaningful descriptions are the basis for all further analysis. Instead of just *naming* vulnerabilities we need to understand them. What are their exact properties and side conditions? What are the properties of a system, subsystem or function that has the specific kind of vulnerability vs. those of one that does not? What are the exact symptoms of the vulnerability or, preferably, a class of vulnerabilities, and how can these symptoms be observed or tested? What is its exact impact if ex-

ploited, in which context? There have been some attempts in this direction already [17,19] but such knowledge ages.

Once we really understand individual vulnerabilities we should attempt to classify them *according to the needs of testing*. This means that we should develop abstractions based on properties that matter for testing, hence the term *symptoms* in the paragraph above. The various types of injection vulnerabilities for instance represent a common underlying problem and should be treated accordingly. Useful abstractions are also needed regarding side conditions. SQL injection for instance if often perceived as a common problem of Web applications while it really is a common problem of database front-ends.

3.2. Causes of Vulnerabilities and the Role of Architecture

It may not always be the programmer who is at fault. Understanding the causes of vulnerabilities will enable us to target our testing. It is well-known that the technologies and platforms used determine, at least in part, the vulnerabilities to be expected. On the one hand a platform can relieve the programmer from certain worries. One example is the Java platform that makes it difficult for the programmer to create buffer overflow errors. On the other hand there are platforms that seem to invite certain types of errors, like Web technologies do for the OWASP Top 10 set of bug types. There may be a simpler explanation here, however, as Web technologies by means of accessibility also invite large numbers of inexperienced programmers.

One particular problem is the role of system architectures. Web technologies, to stick with this example, are based on an architecture that may explain some of the vulnerabilities commonly found in applications based upon them: the developer works on both sides of the trust boundary between client and server but receives little support in observing it. Can we devise a broader understanding of the role of software architectures for software security? We do have some stubs already, such as the principle of least privilege.

Furthermore we must look beyond the limits of the individual system or component at hand and consider its environment(s). It is just too easy to make false assumptions here, so security testing must take into account how the test target is or may be embedded.

The underlying objective is to gain priorities for test planning and stopping or assessment criteria for testing or test results.

3.3. Empirical and Probabilistic Research

We may need more data. Useful data, to be precise, which means data within the framework of appropriate descriptions, classifications and abstractions outlined above. Can we find common principles, laws, or distributions that help us to guide our testing or to interpret its results? Locality principles may be a starting point, as they have been shown to exist for programming errors in general and thus are likely to apply to security bugs too. It would be interesting to see if there are relationships between different types of vulnerabilities, or maybe even between different aspects of software quality.

4. Understanding Security Testing

A theory of security testing must go further than just understanding vulnerabilities. Based on a thorough understanding of vulnerabilities, a theory of testing must also take into account the requirements and constraints of practical testing. The first and foremost research problem is therefore to gather and understand these requirements and constraints. This is what putting practice into theory is supposed to mean, to derive the design of the theory (and later, tools) from the needs of the intended user.

4.1. Field Studies to Determine Requirements

Security testing does exist as a practice. Inferior as it may be, there are white-hat and black-hat hackers, there are in-house and independent labs and there is an academic community that deals with vulnerabilities. It may be worthwhile to have a look at what these communities and individual testers are doing to day, and why they are doing it this way. Lack of better tools may not always be the sole explanation. There are probably personal, economic, cultural and other factors that might be worth studying. Understanding these factors will make it easier to devise techniques and tools that work under the constraints that testers are subject to in their respective environments and communities.

4.2. How is Security Testing Different?

It is commonly felt that security testing is different from other kinds of testing, and this is probably true. However, we do not seem to know yet how exactly it differs, and from what. We do know multiple kinds of testing already; there are worlds between e.g. a unit test and a usability test.

One question therefore is what exactly the differences are, and what their impact is. To what extent can security testing be integrated with other testing activities and the development process in general? Would for instance something like *security test-driven development* make any sense at all or maybe not?

Another question is what we can draw from the various ways of testing that we know today. Usability testing for example, although it seems to target very different issues, may face similar problems as security testing: both attempt to assess an aspect of quality that is difficult to quantify and is best described by the absence of a number of issues. Perhaps we can learn something by analyzing their methods ant the rationale behind.

4.3. Metrics and Quantifications

While metrics are tools, their design needs a sound basis in theory. The question is simple, the answer probably not: are there meaningful metrics and quantifications that help us describe and compare the results of our security testing? Are there other means of lossy yet suitable compression for test results, particularly beyond a table describing the bugs to be fixed? Also, are there better ways than we have today to assess and describe the impact and severity of individual vulnerabilities? Metrics matter in so many ways in business environments that some testers today prefer bad metrics over having none at all, as metrics are something that management understands.

4.4. Analysis and Modeling of Security Requirements

Security testing as such may to an extent be possible without understanding the exact security requirements for a system. The reason is simply that there are generic vulnerabilities whose exploitation will violate any security requirement that one could imagine. In other words, there are a few things that are always wrong.

Generally, however, the results of security testing will be more useful if interpreted with respect to the actual security requirements of a system or component and their tradeoffs with other requirements. Things get complex here again; what is appropriate in one context may be a vulnerability in another. We therefore need means to analyze and represent the security requirements of an existing system as seen by the tester, and means to derive conclusions. Such means should allow us to be accurate where we need to yet sketchy where information is missing. We also need means to analyze trust relationships and other security-critical dependencies in a system.

4.5. Systematic ~~Destruction~~Collection of Evidence

Security testing today still means more or less the same as hacking: a creative activity. Before trying to automate this activity it is worthwhile to ponder it for a while. Particularly, are there more systematic ways of triggering security-critical conditions that go beyond random fuzzing and fault injection?

What might be even more important than systematic destruction – one can always put a system on the Internet as a honeypot and see what happens – are systematic approaches to collecting and interpreting evidence. We know how to fuzz, we know how to inject faults, but how should we interpret the results? Attempting to write an exploit is probably not the most efficient way of doing it.

4.6. The Role of Humans as Testers

There are at least two distinct approaches to security testing, or really to any testing. One approach is to aim for the largest possible amount of automation. As regards security, this approach has failed so far, except for very few niches. For comprehensive security testing – as opposed to testing for very specific types of vulnerabilities – the state of automation as applied in the real world is marked by Web application vulnerability scanners and their poor performance [14,15].

The other approach is to aim for *support* rather than automation. This approach assumes that the human remains in a central position as a tester, and that the primary purpose of tools is to support him in this role.

There is no need to pick one of these approaches and follow it exclusively. However, designers of methods and techniques should be aware that these – and possibly more – different approaches exist, and that automation is not necessarily the best and only way to go.

Experience in the field suggests that human creativity is as invaluable in security testing as it is in attacking systems, and that the most successful and most useful tools are those that support common tasks of testers. Interactive proxies for Web application testing, such as WebScarab [7], are one example. A systematic design of tools of this kind, based on a better understanding of the testers' task and needs, may produce better results than aims for automation that fail in practice due to their requirements.

On the other hand, to the author’s knowledge we do not have any evidence regarding which approach might

be more useful. Empirical research is needed here again, as outlined in the next subsection.

4.7. Meta-testing: Testing Test Tools

Empirical research is needed about the performance of test tools and testing technique. This probably requires, as a prerequisite, designing specific research methods. Obvious issues to overcome are the impact of the individual tester if a method or tool is not fully automated, and the distinction between actual testing and auxiliary activities, such as crawling a Web application for URLs to be tested. In order to improve the technique one needs to know where and how exactly it fails.

4.8. Aspect-oriented Testing?

Security has been considered as an aspect, a cross-cutting concern, in the sense of aspect-oriented programming (AOP). The focus seems to have been, so far, on typical security functions and features viewed and treated as an aspect during development.

Can we devise new testing methods from a similar point of view? This seems justified by the fact that a vulnerability *anywhere* in a system can potentially have an impact on security (although a system should be designed in such a way that it won't). Perhaps a conceptual model derived from ideas of AOP leads to new insights for testing.

Such an approach might start from a suitable decomposition of a program or system along with an understanding of what needs to be achieved for the system to be secure in some particular sense. This would be the aspect view, specifying concerns for the program under test. The tester could then attempt to gather evidence whether the concern is properly dealt with by the program as a whole or, after decomposing it into components, by individual parts of it.

In a way, the STRIDE method [13] may be considered an early and limited version of such an approach: it takes a set of generic attack techniques and attempts to analyze how well a system and its components defend themselves against these.

5. Better Tools and Techniques

5.1. Two Examples

Good tools and techniques are simple and universal. Examples that exist already are attack trees and the STRIDE method. These two methods cover one particular aspect of security testing, threat modeling.

Attack trees [19] are a way of thinking about possible attacks against a system. What they lack entirely is a systematic method of developing the tree. However, once an attack tree has been devised in whichever way, it does a good job communicating the threat model and supports analysis. Attack trees make no limiting assumptions; they can be applied to just about any security problem. Yet they are open to formalization [14].

The STRIDE method [13], the name being an acronym for spoofing, tampering, repudiation, information disclosure, denial of service, and elevation of privilege, is another tool that seems to work in practice. It may be not as universal as attack trees. However, it has the advantage of guiding the tester towards the most important considerations.

5.2. Tools We Need

All the theories are useless if they are not used to create better tools for the tester, be it software tools, processes, techniques or others. Given the fact that security is about preventing intelligent adversaries from reaching their objective, useful tools will likely support human testers rather than trying to replace them (but this is yet another open question: how much automation is good for security testing?)

The call for tools isn't new at all. It can be found almost verbatim, although more concise e.g. in [20]. But not much seems to have happened since, except for the fact that the exciting new tools of then became the standard tools of today.

This subsection is organized as list of keywords with little rationale for individual items. The list is necessarily incomplete. Among others we need:

- Tools and techniques for test planning, that help us identify, for a given system or component, what to look for, where to look, and what the priorities should be.
- Tools that help us deal with complexity and size. Fuzzers and spiders are existing examples: they help us explore input spaces that may be too large for comprehensive testing. Other potentially large spaces are the configurations of a system and its possible environments.
- Approaches to testing that are neither based in hacking nor in formal methods. Hacking is too artistic and often wastes time with obstacles that are easy to overcome in a test lab environment, whereas formal methods are often unsuitable in real-world testing situations that don't meet their particular requirements (e.g. availability of certain knowledge, specifications etc.)

- Tools that work in real settings for various analysis tasks, particularly with incomplete information, such as:
  - Determine and describe attacks that a given security feature does not prevent
  - Requirements analysis for given systems or components
  - Impact analysis for vulnerabilities found
  - Assessing the expected overall number of vulnerabilities based on what has been found
  - The architecture of a system and the interdependencies between its components or between its components and the environment.
  - Analyze non-technical aspects such as usability or economic factors
- Tools and techniques that can be used at early stages of the development process. Such tools should help identify potential vulnerabilities when at the time they are introduced.
- Advanced threat modeling. Needed are techniques that are as easy to use and understand as e.g. attack trees, yet are more powerful. In particular they need to be able to work on incomplete information, such as a set of test results, and provide useful abstractions, such as the power to express e.g. the gains of an attack that achieves an intermediate goal.
- Tools that help observe and model the behavior of systems in such a way that conclusions can be drawn about their security or insecurity.
- Test suites for tools and techniques, and other practical means of evaluating the performance of testing methods.
- Advanced debugging techniques and tools. While the primary purpose of a traditional debugger is to enable the developer to follow the details of execution, debugging-style techniques for security testing should focus on helping the tester to understand the dynamics of an unknown system. For instance a tester might need to know what exactly will be the result of multiple levels of encoding and decoding, or of code generating other code to be executed elsewhere after further transformations. Though it may be wise to avoid such programming altogether, the tester has no choice when encountering a system.
- Better ways of learning, as a community, from experience. Mailing lists such as Bugtraq [1] and Full Disclosure [3] or the CVE archives [2] are of little value as analysis is not as thorough as necessary there.
- Better education for security testers, particularly better than the *trying to become a hacker* approach

And, of course, any tools that emerge from solutions to one of the research problems mentioned above. Note that not all items in above list are mutually exclusive.

All tools and techniques, even automated ones, should be designed on the basis of good usability engineering practice: starting from an understanding of the testers' task and supporting it.

Tools and techniques are unlikely to be successful if they impose too much of an overhead upon the tester, be it in terms of advance learning, of modeling, of configuration, or of results processing. For a tool to be successful it should be accessible to anybody with a basic understanding of software engineering and secure programming.

6. Conclusion

This paper outlines a number of research problems to be addressed in security testing. Some or even all of these problems may have been worked on already by researchers. However, this paper argues that the impact of research upon industry practice seems very limited so far. The author hopes that working on these problems will lead to a better understanding of vulnerabilities and their causes, a better understanding of security testing as such, and to better tools and techniques that work in real-world settings.

There may be further problems and entirely different points of view that have not been considered here. This paper, though trying to outline a research agenda, is meant primarily as an invitation to and starting point for discussions. The key proposal of this paper is to start from what security testers do today, and improve on it.

A subject deliberately ignored in this paper is testing techniques for specific vulnerabilities and technologies, such as cryptography. In some specialized fields the situation may be different from what is described here. But security testing has to take into account all aspects that matter to the real-world tester, not just idealized showcases.

References

[1] *Bugtraq* mailing list archive, URL: http://www.securityfocus.com/archive/1

[2] Common Vulnerabilities and Exposures, http://cve.mitre.org/

[3] *Full Disclosure* mailing list archive, http://lists.netsys.com/pipermail/full-disclosure/

[4] Nessus, URL: http://www.nessus.org/

[5] The GNU Netcat project, URL: http://netcat.sourceforge.net/

[6] OWASP Top Ten Project, URL: http://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

[7] OWASP WebScarab Project, URL: http://www.owasp.org/index.php/Category:OWASP_WebScarab_Project

[8] Scapy, URL: http://www.secdev.org/projects/scapy/

[9] Socat – multipurpose relay, URL: http://www.dest-unreach.org/socat/

[10] Validation report Xerox WorkCentre/WorkCentre Pro 232/238/245/255/265/275 Multifunction Systems, CCEVS-VR-06-0021, Version 1.0, 6 April 2006, URL: http://www.office.xerox.com/latest/SECCR-04.PDF

[11] Anderson, R. J., *Security Engineering: A Guide to Building Dependable Distributed Systems*, Wiley, New York, 2001.

[12] Anderson, J. R., "Why Cryptosystems Fail", in: *Proceedings of the 1st ACM conference on Computer and communications security*, Fairfax, Virginia, 1993.

[13] Howard, M., LeBlanc, D.C., *Writing Secure Code*. Microsoft Press, 2002.

[14] Mauw, S.; Oostdijk, M., "Foundations of Attack Trees" in: Dongho Won and Seungjoo Kim, editors, *International Conference on Information Security and Cryptology – ICISC 2005*, LNCS 3935, pages 186-198, Seoul, Korea, December 2005. Springer-Verlag, Berlin.

[15] McGraw, G., *Software Security: Building Security In*, Addison-Wesley, 2006.

[16] Miller, B. et al., "Fuzz Revisited: A Re-examination of the Reliability of UNIX Utilities and Services", Technical Report CS-TR-95-1268, University of Wisconsin, April 1995.

[17] O'Connor, B., "Vulnerabilities in Not-So Embedded Systems", Black Hat USA 2006 presentation slides, 2006. URL: http://www.blackhat.com/presentations/bh-usa-06/BH-US-06-OConnor.pdf

[18] Peine, H.; Mandel, S., "Sicherheitsprüfwerkzeuge für Web-Anwendungen", IESE-Report, 048.06/D, Kaiserslautern, 2006.

[19] Schneier, B., "Attack Trees: Modeling security threats", *Dr. Dobb's Journal*, December 1999.

[20] Thompson, H. H., "Why Security Testing is Hard", *IEEE Security & Privacy*, vol. 1, no. 4, July-August 2003.

[21] Thompson, H. H., Whittacker, J. A., Mottay, F. E.. "Software Security Vulnerability Testing in Hostile Environments", in: *Proceedings of the 2002 ACM Symposium on Applied Computing (SAC)*, March 10-14, 2002, Madrid, Spain. ACM 2002.

[22] Whittacker, J. A.; Thompson, H. H., *How to Break Software Security*, Addison-Wesley Longman, Amsterdam, 2003.

[23] Wiegenstein, A; Weidemann, F; Schumacher, M; Schinzel, S, "Web Application Vulnerability Scanners - a Benchmark", Version 1.0, Virtual Forge GmbH, 2006-10-04. URL: http://www.virtualforge.de/whitepapers/web_scanner_benchmark.pdf

[24] Yee, G., "Recent Research in Secure Software", NRC Paper No.: NRC 48478; ERB-1134, National Research Council Canada, March 2006.