MEFORMA Security Evaluation Methodology
A Case Study

Ernő Jeges, Balázs Berkes, Balázs Kiss and Gergely Eberhardt
SEARCH-LAB Security Evaluation Analysis and Research Laboratory, Budafoki út 91/C, Budapest, Hungary

Keywords: Security Evaluation Methodology, Security Testing, Security Objectives, Threat Modelling, Case Study, Embedded Systems Security.

Abstract: Even software engineers tend to forget about the fact that the burden of the security incidents we experience today stem from defects in the code – actually bugs – committed by them. Constrained by resources, many software vendors ignore security entirely until they face an incident, or are tackling security just by focusing on the options they think to be the cheapest – which usually means post-incident patching and automatic updates. Security, however, should be applied holistically, and should be interwoven into the entire product development lifecycle. Eliminating security problems is challenging, however; while engineers have to be vigilant and find every single bug in the code to make a product secure, an attacker only has to find a single remaining vulnerability to exploit it and take control of the entire system. This is why security evaluation is so different from functional testing, and why it needs to be performed by a well-prepared security expert. In this paper we will tackle the challenge of security testing, and introduce our methodology for evaluating the security of IT products – MEFORMA was specifically created as a framework for commercial security evaluations, and has already been proven in more than 50 projects over twelve years.

1 INTRODUCTION

Hacking is not just an arcane activity committed by social outcasts with a strange hobby – not anymore (Kirwan, 2012). There are well-organized criminals who are making good money by taking over computers through attacking vulnerable software across the Internet, and creating botnets consisting of millions of zombie machines to do their bidding. Hacking became a big business (Moore, 2009), an industry on its own within the area of cyber-crime.

Due to the effective financial motivation, attackers are coming up with newer attack methods literally every day. The attack landscape is continuously changing; new technologies appear regularly, usually solving some known problems, but – most of the time – they also introduce new ones. We very rarely have silver bullet solutions to problems (Brooks, 1986), and trying to keep up with attackers is an eternal cat-and-mouse game.

The landscape of motivations also changes. From cyber-crime we are apparently moving towards cyber-war and cyber-terrorism (Andress, 2011), with major governmental players expending massive resources, resulting in much more severe consequences that we are yet to experience. Stuxnet (Symantec, 2010), Duqu, Flame and Gauss (Bencsáth, 2012) are examples of complex malware developed by security specialists for millions of dollars that can destroy factory machinery or spy on targeted victims while being undetectable for long enough to do their job. Yet even these are doing nothing else but exploiting security vulnerabilities – actually: bugs – being present in software products.

So vulnerabilities are here to stay. But several sources confirm – including CERT (CERT Software Engineering Institute, 2010), SANS (Walther, 2004), Gartner or Microsoft – that around 90% of attacks do actually exploit well-known vulnerabilities, which have been already published at least six months before the attack took place. Thus, usually we have solutions for the occurring problems, but we just don't use them; it is similar to driving cars without the safety belts fastened.

Securing software is possible in many different ways. Considering the product development lifecycle, approaches may consist of educating engineers about secure coding practices, doing security by design, appropriate implementation (i.e. coding), accomplishing security testing of the

Jeges E., Berkes B., Kiss B. and Eberhardt G..
MEFORMA Security Evaluation Methodology - A Case Study.
DOI: 10.5220/0004919902670274
In *Proceedings of the 4th International Conference on Pervasive and Embedded Computing and Communication Systems* (MeSeCCS-2014), pages 267-274
ISBN: 978-989-758-000-0

product, secure deployment and operation, and many more. Not all approaches are generally equivalent regarding efficiency or usefulness, and each development team should analyse the best return of investment for each; a good source of wisdom of the crowd in this area is the Building Security In Maturity model (BSIMM, 2013), a survey intended to collect and document possible activities that companies are applying to secure their products.

1.1 The Nature of Security Evaluation

Security evaluation of products is a challenging discipline that requires a fundamentally different mind-set from functional testing. While functional testing simply consists of the verification of well-defined requirements and the goal is usually to statistically decrease the number of bugs, security testing involves finding evidence of abnormal operation in non-obvious borderline cases: it is not about how the system should work (aligned to the various use cases), but rather about how it should not (considering misuse and abuse cases).

Since during security testing we are looking for possible behaviour outside of the specified functionality, in theory we should check "anything else", which is obviously an infinite and hard-to-define set. This is why – due to the bounded resources – one should first prioritise by doing a risk analysis. To do that, solid expertise in security is needed, complemented with the ability to think as the attackers would do, and be aware not only of the applicable methodologies, techniques and tools, but also of the trends among the attackers.

1.2 Security Evaluation Methodologies

Various methodologies exist that aim at security evaluation of products – some of them general, some of them domain-specific.

One of the best-known schemes for certification of products from a security point of view is Common Criteria, applying the Common Methodology for Information Technology Security Evaluation (CEM, 2012). The methodology defines roles and the evaluation process itself, distinguishing between evaluating a Protection Profile (describing generic criteria for certain product family) or an actual product as the Target of Evaluation (ToE). A Common Criteria evaluation, however, needs enormous resources, which are often not in line with the evaluated product.

Microsoft has gone a long way in the last decade to secure its products, in parallel defining a methodology for secure software development. Microsoft Secure Development Lifecycle (MS SDL, http://www.microsoft.com/security/sdl/default.aspx) is a process covering all steps of software product development; however, there is only a weak focus on testing, since its Verification step includes only the practices of applying dynamic analysis, fuzz testing and the review of the attack surface.

One of the most referred sources regarding security in the Web application domain is the OWASP – Open Web Application Security Project (http://www.owasp.org). One of its sub-projects is the Application Security Verification Standard (ASVS, 2013), which defines a methodology for assessing the security level of products, governed by a guidance and with a commercial approach through providing requirements for project contracts. ASVS defines four levels of verification: 0-Cursory, 1-Opportunistic, 2-Standard and 3-Advanced, along with verification requirements for each; though parts of the methodology are stated quite generally, its focus is still solely on Web applications.

Open Source Security Testing Methodology Manual (OSTMM, 2010) provides a quite general approach, and is a continuously developing, peer-reviewed – open community developed – methodology based on verified facts. The overall process is, however, at some points quite vague, and some find it hard to translate the presented general approach to actual evaluation steps and procedures.

Some of the above-described methodologies are too domain-specific, yet some are incomplete considering the whole evaluation process or are quite vague in describing certain steps. In the following Chapter we will introduce our comprehensive, practice oriented approach that has a proven track record in evaluating the security of various IT products.

2 MEFORMA OVERVIEW

MEFORMA is a security evaluation methodology designed to be customer-oriented, meaning that the evaluations are being accomplished on a project basis using up resources fixed in advance, and the outcomes not only provide a passed-or-failed result like most of the certification schemes, but by the recommendations given, the development groups also receive valuable support on how to correct the found problems.

Aligned to the usual terminology, ToE denotes the system being evaluated, and we have two simple

roles, the Developer and the Evaluator.

The project approach implies that the work is accomplished in different phases that build upon each other, and that each phase ends with providing a deliverable documenting the results. A typical MEFORMA evaluation project consists of the following phases:

- **Preparation Phase:** The test environment is established, and threat modelling is performed on the ToE – based on its results, test cases are specified. Deliverable is the ***Evaluation Plan*** that contains the definition of the scope, the identified security objectives, the threat model, and the test cases.
- **Evaluation Phase:** The defined test cases are executed, confirming whether the originally identified threats are viable or not. Verified threats (findings) are reported to the Developer regularly through ***Weekly Status Reports*** documenting the progress of the evaluation.
- **Documentation Phase:** The findings are collected, all threats are enlisted and a risk analysis is performed. Most importantly, recommendations are given to the Developer explaining how it should deal with each threat. All results are compiled in the ***Evaluation Report*** as the main deliverable of the project.
- **Review Phase:** During this final phase, a new and fixed version of the ToE is re-evaluated (regression testing) to determine whether the identified threats had been adequately addressed. Evaluation Report is updated with the new results, forming the ***Review Report***.

In the following chapter we introduce the steps of the methodology in more detail.

3 THE EVALUATION PROCESS

In the subsequent sections we will explain the steps of the evaluation. The Preparation Phase (see previous Chapter) consists of Scoping, Information Gathering, Threat Modelling, and finally the Test Case Specification, all documented in the Test Plan. The actual Evaluation takes place based on the agreed Test Plan, followed by the Documentation Phase during which the findings are documented and recommendations are given, and the verified threats are rated by a Risk Analysis. Finally, a low-intensity Review phase allows for the clarification of findings and regression testing.

3.1 Scoping

As the first step, the ToE must be identified, and the scope of the evaluation must be specified, which is a co-operative effort between the Evaluator and the Developer. For this, it is important to fix the ToE (platform, versions, builds, etc.) before the work starts, since the evaluation involves consecutive tests that build upon the results of previous ones.

Basically there are three main aspects of planning an audit: scope, depth of analysis and the audit risk. If we limit the scope of the evaluation, important issues may not be addressed even if we increase the depth of the analysis. Contrarily, if we want to keep the scope as wide as possible and evaluate the whole system, limits in the available resources will imply limitations in the depth of the analysis. In both cases it is important to be aware of those remaining risk factors that the investigations would not cover.

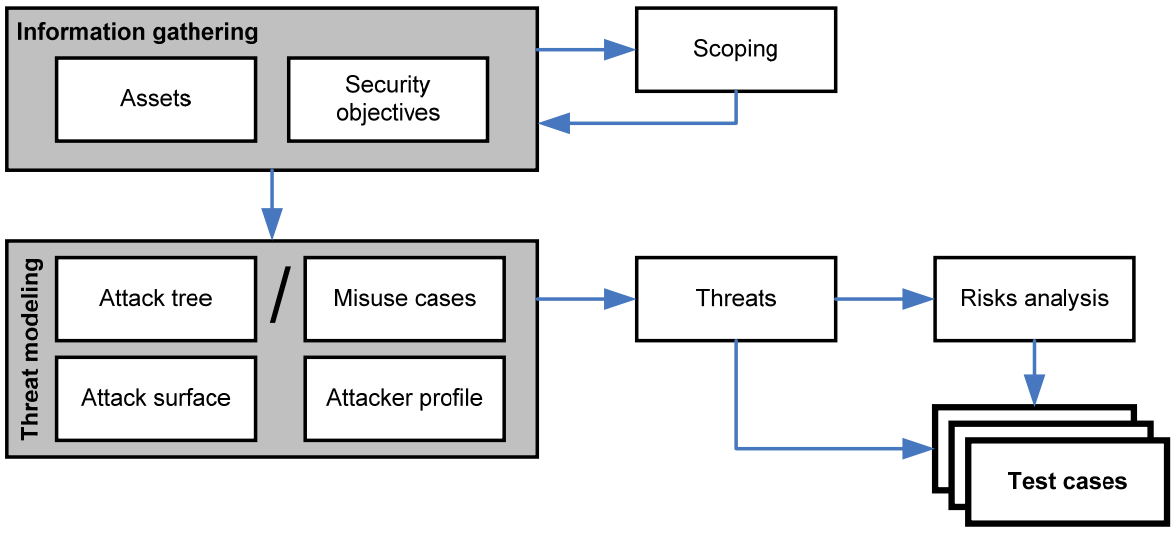

Figure 1: Preparatory steps for the MEFORMA security evaluation.

3.2 Information Gathering

The main goal of this step is to specify the security objectives. For this, one should first identify and understand the assets within the system that need to be protected, and then for each asset determine which of the independent security objectives (typically taken from the CIA triad i.e. Confidentiality, Integrity, and Availability) are relevant. The assets are then further grouped into categories appropriate to the specific evaluation – for instance, a software evaluation would likely have 'software assets', 'data assets', and 'other assets' categories.

If the Developer provides their own list of security requirements, those are used to refine these further, prioritizing the security of the assets the Developer finds to be the most important.

3.3 Threat Modelling

Once the elements of the system are identified and understood, threat modelling is accomplished according to the security objectives. This can be done by various means and following various approaches; we mostly use the following:

- **Attack tree** (Schneier, 1999) modelling consists of conceptual diagrams of perceived threats to a system. This process is performed by identifying an attacker goal, and then modelling the various ways these goals can be achieved.
- **Misuse cases** are similar to the use case UML formalism, but instead of describing ways to use system functionality, they present ways on how to misuse it (Alexander, 2002). The misuse cases are derived from the normal use cases: for instance, for a normal 'shutdown' use case there may be several misuse cases defined where the system can be shut down.

In threat modelling, as security testing is about how a system should not work, there is a remarkable emphasis on the evaluators' security experience, i.e. its knowledge about how things can go wrong, and also about the attack trends among the attackers. For this, one can rely not only on its experience, but can also refer to repositories and knowledge bases, like the SVRS (Security Repository Vulnerability Services, browsable and accessible after registration at https://svrs.shields-project.eu) or the ENISA Threat landscape (ENISA, 2012).

Optionally, attacker profiling may also be performed in this stage. This identifies several different types of attackers that may have different goals when it comes to attacking the security of the ToE, and may also have different resources and expertise at their disposal. Example profiles are insider, exploiter, misuser, and thief.

The end result of this step is a set of threats.

3.4 Test Case Specification

During threat modelling, many potential threats will be identified. Some of these threats may be considered out of scope for the evaluation due to some reasons (for instance being unlikely or out of the control of the Developer, like the vendor's secret key being leaked) or trivial (e.g. a particular aspect of the system is insecure by design). However, most of the threats will require investigation to confirm their feasibility – which is the main goal of the evaluation.

To that end, the evaluators categorize the relevant threats, and accomplish a preliminary risk analysis to reveal which are the most important threats. This prioritization is especially important, since the project size and the allocated efforts might not be in line with the volume of the actually revealed and relevant threats; risk analysis will show which are the issues that should be put in focus and included in the evaluation, and which will be possibly omitted due to lack of resources.

As the final step of the preparatory work, test cases are defined, aligned to the threats in focus. By accomplishing these test cases, we can check if the associated threats are real, i.e. if it is feasible to execute an attack and realize the threat.

Results of Scoping, Information Gathering, Threat Modelling and Test Case Specification are all summarized in the Test Plan, which is refined and agreed with the Developer through several iterations.

3.5 Evaluation

Building upon good preparatory work, the evaluation simply means the execution of the test cases already specified. Actual evaluation of a test case can consist of black-box / white-box / grey-box (Kicillof, 2007) testing or source code review, and can also include reverse-engineering of the system. Manual execution of test cases can be mixed with automated evaluation, for instance fuzzing (Miller, 1990) or penetration testing tools.

The goal is basically to determine if any of the identified threats can be realized through any flaw or vulnerability that exist the system.

3.6 Documentation and Recommendations

As a result of the evaluation a list of verified threats is compiled, i.e. the *findings*. Most of the time, these are threats that were already identified during the threat modelling step, but completely new threats can also surface during the evaluation.

When documenting the threats, we use several symbols to denote the results of individual tests within a test case. These symbols are as follows:

- ✓ **Normal operation.** The outcome of the test indicates that the implementation is correct; no findings.
- 💣 **Problem.** The outcome of the test has clearly identified a security problem.
- 😐 **Potential / possible problem.** The outcome of the test does not clearly indicate a security problem, but may lead to unexpected or abnormal operation. This symbol is also used if a security issue is suspected, but could not be verified.

Figure 2: Documenting the findings.

For each verified threat, recommendations are given for techniques that could be used to completely eliminate the threat, or at least reduce the associated risks. In each case, the goal is to reduce either the likelihood or the severity of the threat.

All results are contained in an Evaluation Report delivered to the Developer, who should make the appropriate steps to address the found issues.

3.7 Risk Analysis

We estimate the risk each discovered threat poses to the system. This is done by specifying the severity (damage that can be potentially done by realizing the threat) and likelihood (the difficulty of realizing the threat) of each threat. This latter can be dependent of many factors, including the needed resources (time, money, tool accessibility, etc.) and the expertise level required by an attacker to realize the threat, i.e. commit a relevant attack.

The risk is the product of these two values using the standard likelihood X severity risk calculation:

Table 1: Likelihood X severity risk calculation.

Likelihood / Severity	Low	Medium	High
Low	Very Low	Low	Medium
Medium	Low	High	Very High
High	Medium	Very High	Catastrophic

The risk value of each threat can take the following levels:

- **Very Low (VL):** The threat has a very minor – but still not negligible – effect on the security of the asset.
- **Low (L):** The threat has a minor effect on the security of the asset.
- **Medium (M):** The threat has a noticeable effect on the security of the asset.
- **High (H):** The threat significantly endangers the asset.
- **Very high (VH):** The threat significantly endangers the asset or the system as a whole.
- **Catastrophic (C):** The threat presents a critical risk to the system as a whole; if not mitigated, its effects could put the entire business process at risk.

3.8 Review

Following the evaluation, a several weeks long review phase is usually reserved for the Developer to review the results and fix the revealed weaknesses. At the end of this period, the Evaluator receives a new version of the ToE, and re-runs the relevant test cases on it to verify if the threats have been appropriately tackled.

A residual risk analysis is accomplished showing the threats that still remain in the system after the review. Any new threats discovered during the review are also presented in the Review Report.

4 A CASE STUDY

The case study evaluation described in this section was performed as part of the nSHIELD project (see acknowledgements).

4.1 The ToE

The Target of Evaluation was an integrated secure

node prototype running on an ARM-based development board. The protection of node functionality was realized through two main software components: a **secure boot loader** and a **hypervisor**.

The secure boot loader is the code that is first run on the ToE. It ensures the integrity of the firmware that is loaded and run afterwards, and which also contains the hypervisor and the protected application running on the board.

The hypervisor guarantees the isolation of applications running on the platform as well as secure interaction between trusted applications and open components. An example trusted application running within the hypervisor was also provided for the evaluation.

As the first step, we had to define the scope of the evaluation. Considering the dependencies between the two components – specifically, the hypervisor's dependency on the secure boot loader to maintain integrity within the node – we decided to perform a combined evaluation of the two software components described above.

After the scope definition, we identified the security objectives that were relevant for the ToE. We focused mainly on integrity objectives from the CIA triad model, since – aside from the trusted application and its cryptographic assets – confidentiality and availability objectives were not in the scope. We identified the following assets:

- **Hardware assets:** secure flash, persistent storage
- **Software assets:** secure boot loader, hypervisor, trusted application, kernel
- **Cryptographic assets:** trusted application keys, secure boot loader public key

We then identified the main threats that an attacker may need to realize in order to achieve their goal (compromising the node's operation). These threats consisted of **bypassing the bootloader's integrity protection**, **obtaining confidential data from the trusted application**, and **breaking out of the virtual guest mode** provided by the hypervisor.

Based on the identified threats, we have specified the following test cases in the Test Plan:

- *Source code analysis of the secure boot loader* – checking whether the main function and high-level logic of the secure boot loader were implemented correctly.
- *Security of the signature verification process* – checking whether image validation was implemented correctly, and whether the process itself was free of cryptographic issues, e.g. the Bleichenbacher attack (Kühn, 2008).
- *Source code analysis of other libraries* – checking whether additional libraries used by the secure boot loader to manage hardware devices (MMC, VFAT, etc.) were free of typical security flaws.
- *Source code analysis of hypercalls* – checking whether the implementation of hypercall handlers were free of typical security flaws.
- *Protection of virtual guest modes* – checking whether the state of the virtual guest mode was protected and whether the implementation was free from any logical flaws.
- *Protection of trusted application* – verifying whether it was possible to bypass the Hypervisor and access the trusted application in any way.

4.2 Evaluation Results

As we had access to the source code of the secure boot loader and hypervisor components, source code analysis was the main approach to follow during the evaluation. Additionally, for the *Source code analysis of hypercalls* test case, we set up a runtime environment to verify our results.

During the evaluation, we have identified and verified several threats that could impact the security of the prototype. These are listed below; along with the appropriate recommendations, these were communicated to the Developer in the Test Report.

- *Integrity protection could be bypassed* – Due to a design flaw and several implementation flaws, it was possible to bypass the integrity protection features of the secure boot loader.
- *Security-critical programming errors* – Several programming bugs in the prototype could be potentially exploited by an attacker to crash the node, or – in some cases – even take control of it.
- *Signature padding was not checked properly* – The secure boot loader's signature checking function did not check the padding at the end of the signature properly; this was a Bleichenbacher-type weakness, but was not exploitable due to the RSA exponent used.
- *Library contained known vulnerability* – An old version of the PuTTY library (http://www.chiark.greenend.org.uk/~sgtatham/putty/) was used for RSA operations; this version contained a known buffer overflow vulnerability that could cause a crash or even execution of malicious code.
- *Hypervisor-level code execution possibility* – A Write-What-Where vulnerability in the hypervisor could allow an attacker to write arbitrary data into any place in memory.
- *Guest privilege escalation possibility* – A logic error in hypercall handling could allow a guest to

change its mode to trusted; however, the hypervisor would crash soon thereafter.

- *Weaknesses in the trusted application* – Several recurring programming errors and design issues were found in the example trusted application that could allow an attacker to obtain the application's secret data as well as forge contract signatures that would be accepted as valid. Even though this example application was meant to be only a proof of concept implementation, it could be used as a basis for developing an example application later. For this reason, this issue was still reported to the Developer in order to make future ToE releases more secure.

After having the above list of verified threats, we performed a risk assessment, the results of which are presented in the next Table ordered by risk value (S: Severity, L: Likelihood, R: Risk – see section 3.7 for more details).

Table 2: Risk assessment for the evaluated prototype.

Threat name	S	L	R
Integrity protection could be bypassed	H	M	**VH**
Security-critical programming errors (secure boot loader)	H	M	**VH**
Weaknesses in the trusted application	H	M	**VH**
Hypervisor-level code execution possibility	VH	L	**H**
Guest privilege escalation possibility	L	H	**M**
Library contained known vulnerability	M	L	**L**
Signature padding was not checked	L	L	**VL**
Security-critical programming errors (hypervisor)	L	L	**VL**

Following the delivery of the Test Report, all of the identified vulnerabilities in the prototype have been addressed by the Developer, and this has been verified by the regression testing accomplished in the Review Phase. The presented case study proved once again that the approach provided by the MEFORMA methodology can be used to successfully improve the overall security level of any system.

5 CONCLUSIONS

The MEFORMA security evaluation methodology was created to provide a framework for commercial security evaluations, and it has been used – and continuously refined – in more than 50 evaluation projects over twelve years, aiming at assessing the security of various software and hardware based products, in the domain of telecom, banking, finance and consumer electronics, among others.

As compared to Common Criteria (CC) which focuses on the development process and the proofs developers should produce during their work, our methodology provides a cost efficient approach in focusing on the actual results of the design and development work, i.e. the security of the actual ToE itself. Thus the primary aim of MEFORMA is to help developers in creating more secure products.

In the presented case study, we have shown the effectiveness of our systematic approach, which revealed 7 weaknesses of the evaluated platform: a hypervisor supported by a secure boot loader. Moreover, based on the recommendations given to address each revealed problem, the Developer has fixed the evaluated system, and the correctness of the fixes has been verified by regression testing in the final Review Phase of the evaluation. After the conclusion of the evaluation, it has resulted in an implementation of the prototype that is much more resistant to various attacks.

For further improvement of the methodology, we plan to do a comprehensive study of the threat modelling landscape, and optionally include some recently introduced techniques in this critical step of the process. In addition, we plan to introduce a suggestion on the tools and techniques (e.g static code analysis or fuzz testing) to be used during the actual evaluation, since this step is at the moment completely dependent on the evaluator's expertise.

ACKNOWLEDGEMENTS

MEFORMA is a security evaluation methodology developed by SEARCH-LAB Security Evaluation Analysis and Research Laboratory for its everyday assessments.

The case study has been accomplished within nSHIELD project (http://www.newshield.eu) co-funded by the ARTEMIS JOINT UNDERTAKING (Sub-programme SP6) focused on the research of SPD (Security, Privacy, Dependability) in the context of Embedded Systems.

REFERENCES

Kirwan, G. and Power, A., 2012. *The Psychology of Cyber Crime: Concepts and Principles*, IGI Global.

Moore, T., Clayton, R., and Anderson, R., 2009. *The Economics of Online Crime*, Journal of Economic

Perspectives, American Economic Association, vol. 23 (3), pp. 3-20.

Brooks, Jr.F.P., 1986. *No Silver Bullet—Essence and Accidents of Software Engineering*, Information Processing 86, Elsevier Science Publishers, pp. 1069-1076.

Andress, J., Winterfeld, S, 2011. *Cyber Warfare: Techniques, Tactics and Tools for Security Practitioners,* Syngress.

Symantec Security Response, 2010. *W32.Stuxnet*. http://www.symantec.com/security_response/writeup.jsp?docid=2010-071400-3123-99; last visited: October 2013.

Bencsáth, B., Pék, G., Buttyán, L., Félegyházi, M., 2012. *The Cousins of Stuxnet: Duqu,* Flame*, and Gauss*, Future Internet 2012, 4, pp. 971-1003.

CERT Software Engineering Institute, 2010. *Software Assurance Curriculum Project Volume I: Master of Software Assurance Reference Curriculum*, Technical Report.

Walther, J., 2004. *Meeting the challenges of automated patch management*, GSEC practical assignment.

BSIMM, *Building Security In Maturity Model*, Release V, October 2013. http://bsimm.com/; last visited October 2013.

CEM, 2012. *Common Methodology for Information Technology Security Evaluation*, v3.1, revision 4. http://www.commoncriteriaportal.org/files/ccfiles/CEMV3.1R4.pdf; last visited: October 2013.

ASVS, 2013. OWASP Application Security Verification Standard 2013, version 2.0 beta, http://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project; last visited October 2013.

OSSTM, 2010. The Open Source Security Testing Methodology Manual , Contemporary Security Testing and Analysis, version 3, ISECOM, http://www.isecom.org/research/osstmm.html; last visited November 2013.

Schneier, B., 1999. *Attack Trees*. Dr. Dobb's Journal, vol. 24, pp. 21 - 29.

Alexander, I., 2002. *Misuse cases: Use cases with hostile intent*, Software, IEEE, Vol. 20, 1, pp. 58-66.

ENISA European Network and Information Security Agency, 2012. *ENISA Threat Landscape, Responding to the Evolving Threat Environment*, September 2012.

Kicillof, N., Grieskamp, W., Tillmann, N., Braberman V. A., 2007. *Achieving both model and code coverage with automated gray-box testing*, in: 3rd Workshop on Advances in Model-Based Testing, A-MOST'07, ACM Press, pp. 1-11.

Miller, B. P., Fredriksen, L. and So, B., 1990. *An Empirical Study of the Reliability of UNIX Utilities*, Communications of the ACM 33, pp. 32-44.

Kühn, U., Pyshkin, A., Tews, E. and Weinmann, R., 2008. *Variants of Bleichenbacher's Low-Exponent Attack on PKCS#1 RSA Signatures*, Proc. Sicherheit 2008, pp. 97-109.