Risk Assessment and Security Testing of Large Scale Networked Systems with RACOMAT

Johannes Viehmann[1], Frank Werner[2]

[1]Fraunhofer FOKUS, Berlin, Germany
Johannes.Viehmann@fokus.fraunhofer.de
[2]Software AG, Darmstadt, Germany
Frank.Werner@softwareag.com

Abstract. Risk management is an important part of the software quality management because security issues can result in big economical losses and even worse legal consequences. While risk assessment as the base for any risk treatment is widely regarded to be important, doing a risk assessment itself remains a challenge especially for complex large scaled networked systems. This paper presents an ongoing case study in which such a system is assessed. In order to deal with the challenges from that case study, the RACOMAT method and the RACOMAT tool for compositional risk assessment closely combined with security testing and incident simulation for have been developed with the goal to reach a new level of automation results in risk assessment.

Keywords: Risk assessment · Security testing · Incident simulation

1 Introduction

For software vendors risk assessment is a big challenge due to the steadily increasing complexity of today's industrial software development and rising risk awareness on the customer side. Typically, IT systems and software applications are distributed logically and geographically, and encompass hundreds of installations, servers, and processing nodes. As customers rely on mature and ready-to-use software, products should not expose vulnerabilities, but reflect the state of the art technology and obey security risks or technical risks. Failing to meet customer expectations will result in a loss of customer trust, customer exodus, financial losses, and in many cases in legal consequences and law suits.

On the other hand, the impossibility to analyze and treat every potential security problem in advance is well-known. Any security issue without appropriate safeguards could lead to a considerable damage for the customer, be it its loss of business (e.g. when a successful DoS attack prevents business processes form being pursued), loss of data (due to unauthorized access) or malicious manipulation of business process sequences or activities. The task of risk management is to identify and treat the most critical risks without wasting resources for less severe problems. Within this paper, only the risk assessment part of the risk management process is addressed. More precisely, this paper reports the experiences made during the risk assessment for an industrial large scale software system Command Central.

Risk assessment can be difficult and expensive. It typically depends on the skills and estimates of experts and manual risk assessment can only be performed at a high level of abstraction for large scale systems. Security testing is one risk analysis method that eventually yields objective results. But security testing itself might be hard and expensive, too. Manual testing is itself error prone and again infeasible for large scale systems. Choosing what should be tested and interpreting security test results are not trivial tasks. Indeed, even highly insecure systems can produce lots of correct test verdicts if the “wrong” test cases have been created and executed. Therefore, it makes sense to do Risk Assessment COMbined with Automated Testing, i.e. to use the RACOMAT method and the RACOMAT tool introduced here. RACOMAT has been developed along the case study in order to deal exactly with the challenges of large scale networked systems. Both, the development of RACOMAT and the risk assessment of Command Central are still ongoing.

1.1 The case study

The software under analysis is called Command Central [14] from Software AG, a tool from the webMethods tool suite allowing release managers, infrastructure engineers, system administrators, and operators to perform administrative tasks from a single location. Command Central assist the configuration, management, and monitoring by supporting the following tasks:

- Infrastructure engineers can see at a glance which products and fixes are installed, where they are installed, and compare installations to find discrepancies.
- System administrators can configure environments by using a single web user interface or command-line tool. Maintenance involves minimum effort and risk.
- Release managers can prepare and deploy changes to multiple servers using command-line scripting for simpler, safer lifecycle management.
- Operators can monitor server status and health, as well as start and stop servers from a single location. They can also configure alerts to be sent to them in case of unplanned outages.

Command Central is built on top of Software AG Common Platform, which uses the OSGi (Open Services Gateway Initiative) framework. Product-specific features are in the form of plug-ins.

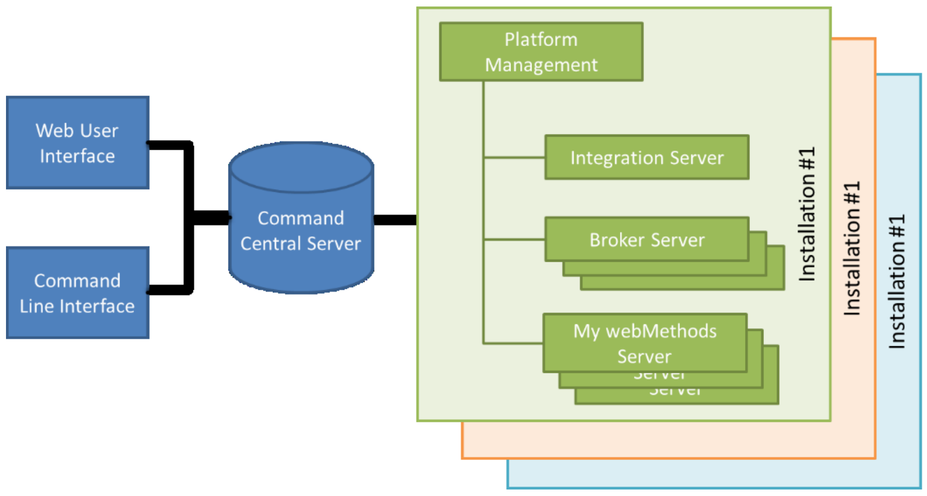


Fig. 1. An installation set-up scenario with Command Central Management

Command Central users can communicate with Command Central Server using either the graphical web user interface for administering products using the web, or the Command line interface for automating administrative operations. An architecture overview of the Command Central software is provided in Fig. 1.

The Command Central Server accepts administrative commands that users submit through one of the user interfaces and directs the commands to the respective Platform Manager for subsequent execution. An installation in Command Central means one or more instances of the products that Command Central can manage. It provides a common location for configuring managed products installed in different environments.

The webMethods Platform Manager manages other Software AG products. Platform Manager enables Command Central to centrally administer the lifecycle of managed products. In a host machine, there might be multiple Software AG product installations. For each Software AG product installation, a separate Platform Manager is needed to manage the installed products.

2 State of the art

Security critical ICT systems should be carefully managed especially with respect to the related security risks. Such a risk management should include well-known concepts like risk assessment (ISO 31000 – [2]) and security testing (ISO 29119 – [3]).

2.1 Risk Assessment

According to the ISO 31000 standard, risk assessment means to identify, analyze and evaluate risks which could damage or destroy assets [2]. Lots of different methods and technologies for risk assessment have evolved, including fault tree analysis (FTA) [5], event tree analysis ETA [6], Failure Mode Effect (and Criticality) Analysis FMEA/FMECA [4] and the CORAS method [1].

Compositional risk assessment allows analysts to deal with manageable small components of a complex large scale modular system. It combines the individual risk assessment results for components to derive a risk picture for the entire complex system without looking further into the details of its components. However, most traditional risk assessment technologies analyze systems as a whole [7]. They do not offer support for compositional risk assessment. Nevertheless there are some publications dealing with compositional risk assessment and suggesting extensions for the mentioned risk assessment concepts, e.g. [8] for FTA and [9] for FMEA or [10] for CORAS, which is used in the case study presented here.

There are huge databases of common weaknesses, attack patterns and safeguards available that can be used as a base for security risk assessment, for example Mitre CWE [23] and CAPEC [22] or BSI IT-Grundschutz [21]. There are also vulnerability databases that list specific vulnerabilities for existing software programs, e.g. Mitre CVE [24]. Such information could be helpful in compositional risk assessment for systems that use listed programs or that have them in their environment. The mentioned catalogues are used in the case study introduced here.

2.2 Security testing and testing combined with risk assessment

The ISO 29119 standard defines security testing as a type of testing that tries to evaluate the protection level of the system under test against unauthorized access, unwanted use and denial of service [3].

Traditional testing is a method to analyze the behavior of a system according to its specified functionality and expected results. Security testing in contrast also tests for unspecified behavior and for unexpected results. Hence, compared to other types of testing, for security testing it is harder to decide what should actually be tested and is more challenging to interpret the observed behavior.

One possible way to deal with the challenges of security testing is to combine it with risk assessment. ISO 29119 defines risk-based testing as a general method that uses risk assessment results to guide and to improve the testing process [3]. Risk analysis results can be used to define test policies and strategies, to identify what should actually be tested and how much effort should be spend for it. For instance, Kloos et al. use fault trees for identifying test cases [15]. Stallbaum and Metzger automated the generation and prioritization of test cases based upon risk assessment artefacts [16]. Even (semi-) automated risk-based security testing might be expensive. Reusing testing artefacts for recurring security testing problems might help to reduce the effort. Security test patterns have been suggested for that purpose [17], but currently there is no extensive library of useful security test patterns available.

Risk assessment and testing can also interact the other way around: in test-based risk assessment, test results are used to identify, analyze and evaluate risks. There are several publications about this approach [18] [19], but there is still no general applicable method and not much tool support.

The concepts of risk-based testing and test-based risk assessment can also be combined. While Erdogan et al. propose such a combination [18], they do not propose any technique or detailed guideline for how to update the risk model based on the test results. The combined method with tool support presented in [20] has been developed further along the case study presented here and it was named the RACOMAT method and tool.

2.3 Simulation

Simulations that work with simplifying models in general can be very helpful to analyze large scale systems. For instance, Monte Carlo simulations can be used to analyze the behavior of complex systems and especially for calculating likelihood values in the risk aggregation process [11] [12].

In [13] it is described how Monte Carlo simulations and security testing can be used together within an iterative risk assessment process in order to refine the risk picture. This approach is used in the case study described here.

3 Requirements, Problems and Expectations

Large scale networked computer systems and software programs can be enormous complex. Building and maintaining such systems is challenging and it can be very expensive. One way to reduce costs without reducing the product features and the product quality is to let as much work as possible be done automatically by machines.

However, in the production and lifecycle management process for software there is usually only limited potential for automation. It typically requires lots of creative work which nowadays has to be done manually, e.g. modelling or writing code directly.

Nevertheless, at least for analytical and recurring tasks in the software development and maintenance process, a high level of automation should be achievable.

These promising candidates for automation include especially testing as a vital part of the software quality management. Indeed, within a testing process, many tasks can be automated, especially test data generation and test case execution. There are lots of tools supporting automated testing. Testing for specified behavior can be more or less completely automated if the specification is well modeled – test cases can be derived automatically from such a model. Of course, appropriate models have to be created manually in the first place. Additionally, interpreting the test results and reacting properly on them will always require further manual work.

For security testing, automation is probably slightly more difficult than for other types of testing like functional testing or integration testing. Security testing requires to test for unspecified behavior, so there is obviously no trivial way to derive all relevant test cases from a specification. Deciding what might be security critical and what should therefore be tested is a very difficult task because complete testing is infeasible for complex systems. Once having decided what should be tested, with testing techniques like fuzzing, it might be possible to automatically generate great many security relevant test cases. But that is only the easier part of security testing. Judging automatically whether a test has passed or failed can be quit challenging. For security tests it is often not at all clear what behavior could be triggered. Observing and interpreting unexpected incidents is indeed tricky. There is no point in generating automatically lots of test cases if all test results could be false positives or false negatives: the required further investigations would probably lead to an amount of manual work, which would be higher than simply doing manual testing with fewer, but eventually better designed test cases.

In contrast to testing, risk assessment is typically done with lots of manual effort. Conventional risk identification and risk analysis heavily depend on expert work and expert judgement. There are tools for risk assessment, but still questioners have to be answered, risk models have to be created and managed – so there is still a lot of work that has to be done more or less manually. Hence, for complex large scale systems, traditional risk assessment can only be performed at a high abstraction level.

We believe that the combination of risk assessment and security testing will lead to a better level of automatization for both concepts. Incident simulations are a third analytical concept that could be used to integrate them into a single process.

Besides automation, component based techniques are another important approach to deal with complexity. Compositional risk Assessment allows to treat small manageable components more or less independent from each other. Results for individual components are composed to a big picture without looking again in the details of the components. This is especially helpful for continuous risk assessment of systems that are gradually updated because it limits the need to reassess the risk to those components that have actually changed.

To be able to conduct efficient risk assessment of newly implemented features is a very appealing feature in software industries. Whenever a new feature is addressed by development, the assessment should be updated in order to make sure it contains a careful analysis of the new or altered product. Risk assessment must continuously assure that the risks are not only identified at lowest component level, but also for the higher component levels and single product levels up to the product suite level. Traces between the same risk artifacts at different abstraction levels should be tracked and preserved.

Risk evaluation should be possible at any level of abstraction. If for example some likelihood value for a risk was determined with the help of security testing, then it should be possible to look into the details of individual test cases and test results. But it should also be possible to view just the top level risks for all the program suites in which the tested component is used. A tool for risk assessment and security testing should support such transparent views. Additionally, it should provide at any level of abstraction information what the individual technical risk artefacts mean for the organizational management, e.g. how they affect business processes and what legal consequences unwanted incidents might have. Aggregated risk assessment results and risk pictures with connections to their non-technical impacts are suitable as a basis for decision making.

At the beginning of the research work for the presented case study, there was no risk assessment tool available that fulfills these expectations of interdisciplinary transparency and seamless integration with security testing.

Software products are implemented according to an intended environment and intended usage scenarios. Risk sensitivity in different environments and scenarios can vary in the sense that the product may be suitable for a particular setting (for which it was initially designed) but exhibits an unacceptable risk which prohibits the use in another, even more critical setting. In our view security should not be seen as a goal in itself, but a means of protecting assets. Cyber security must be understood and reasoned about not just at a technical level, but also at a non-technical level, taking into account the context in which software is used, organizational level assets, and legal issues. The producers of software systems cannot do the complete risk assessment for all potential customers because they do not have insight in the various contexts in which their products may be used. Therefore, software products should be provided with reusable risk assessment artefacts so that the risk of using them in a certain context can be evaluated by the potential customers and users themselves.

4 An integrated Risk Assessment and Security Testing Approach

4.1 Initial Risk Assessment

The work on the case study started according to the ISO 31000 standard with establishing the context and risk identification phases. During these initial phases the product under investigation has been modelled in the ARIS RASEN framework. This has been achieved in a joint workshop with a software engineer as a representative from the product development (Command Central Product Development), a security expert overviewing and ensuring the compliance to security standards, and the RASEN project development team in charge of the implementation. As a result of the workshop the software under consideration has been modelled and weaknesses and risks from the CWE database have been assigned to the product and its components.

These first steps were done manually with the help of an existing risk artefact database. Hence, the initial assessment took place at a high level of abstraction in order to keep the manual effort reasonable low. Nevertheless, lots of information has been collected: for some of the about 30 components up to 27 different potential vulnerabilities have been identified – no less than 11 weaknesses for any component. This initial phase did neither analyze if the weaknesses actually exist nor how likely it is that the existing ones would actually be exploited. Its result is also not detailed enough to be very helpful as a starting point for risk-based security testing. Obviously, further analysis was required.

4.2 Refining the Risk Picture

For a more detailed risk analysis in which automated security testing can be used to get reliable objective results, the RACOMAT method and the RACOMAT tool have been developed. The core of the RACOMAT method is the iterative RACOMAT process (shown in Fig. 1) combines risk assessment and automated security testing in both ways: Test-Based Risk Assessment (TBRA), which tries to improve risk assessment with the results of security tests and Risk-Based Security Testing (RBST), which tries to optimize security testing with results of risk assessment. The method itself is basically following the concepts described in [20].

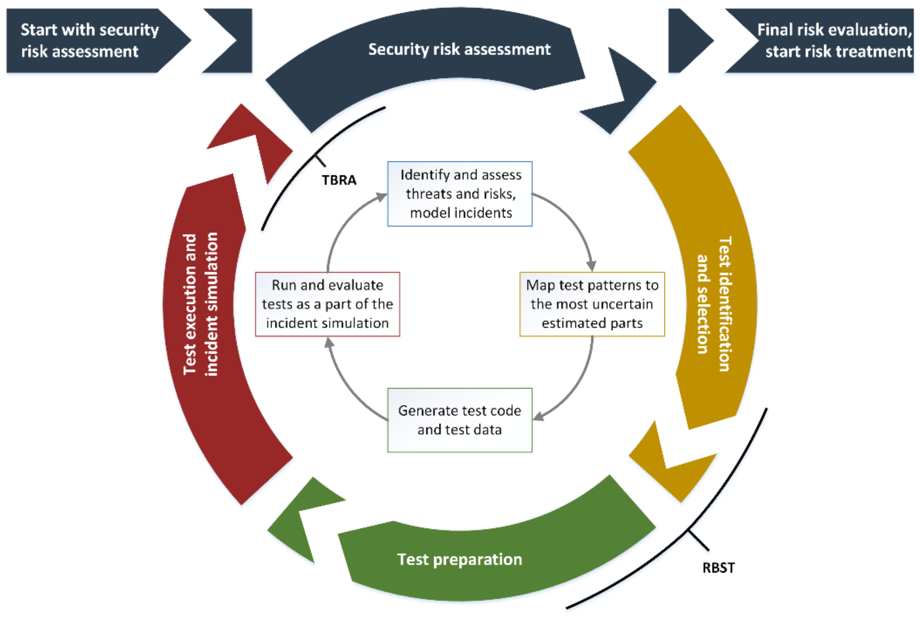

Start with security risk assessment
Security risk assessment
Final risk evaluation, start risk treatment
TBRA
Identify and assess threats and risks, model incidents
Map test patterns to the most uncertain estimated parts
Generate test code and test data
Run and evaluate tests as a part of the incident simulation
Test identification and selection
Test preparation
Test execution and incident simulation
RBST

Fig. 2. The RACOMAT method

The development made along the Command Central case study was basically to improve the applicability for large scale networked systems.

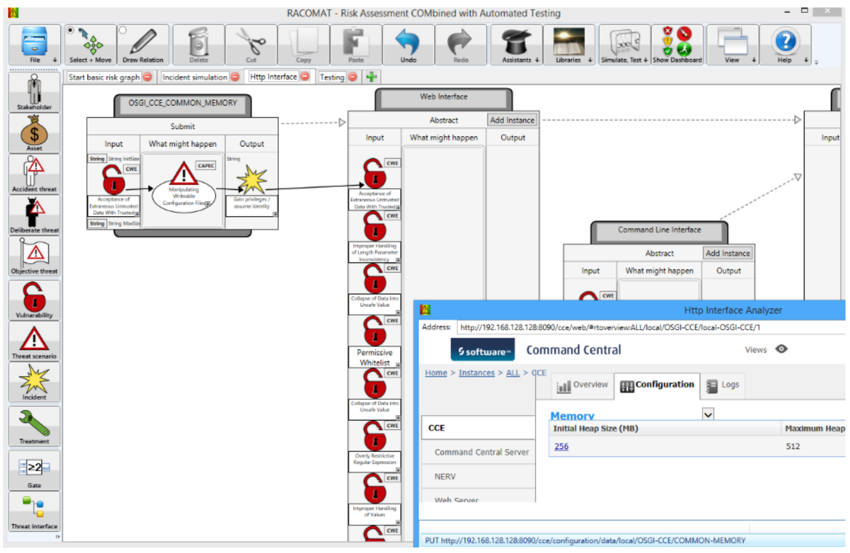
The first idea that came to mind in order to reduce the manual effort of low level risk assessment was to integrate techniques for automated static analysis of components into the RACOMAT tool. Given (X)HTML pages, source program code, compiled libraries or programs, the RACOMAT static analysis tries to identify the public interfaces of any components and especially the functions as well as ports that could be used for interaction with other components or users. Thereby initial system models can be generated without requiring manual actions. The generated threat interfaces contain signatures with input and output ports, which can be associated with risk artefacts. Threat interfaces are low level enough to be used for automated testing, for example.

While the static analysis works fine for some software systems, especially for small exemplary programs, it does not in general produce good results. For Command Central, the current state of the RACOMT tool static analysis proved not to be very helpful. This was a kind of surprising because the main graphical Command Central user interface is a web based user interface and the RACOMAT tool is in general capable to statically analyze HTML user interfaces. However, the way Command Central builds HTML pages containing not much more than lots of script function calls, identifying interface elements is already difficult. Additionally, the user interfaces are generated dynamically based on the session state.

To enable the RACOMAT tool to deal with Command Central and other software systems with highly state dependent interfaces, it has been extended with dynamical analysis features. The dynamical analysis records information while the system is actually used. For the recording process it does not really matter if the system is used by human beings manually or if it is controlled by tools with automated test cases that are executed and monitored. Recording can especially also take place while the RACOMAT tool itself executes security tests.

Technically, the recording can take place at different levels. For a web based application like Command Central, it is probably a good idea to record messages at the Hypertext Transfer Protocol layer. Therefore, a little proxy server has been developed and integrated into the RACOMAT tool. This proxy server can be used to monitor and to manipulate the communication between Web Clients and Web Servers.

The recorded information can be used to generate interface descriptions automatically. For example, the RACOMAT tool can extract the target name and the parameter names from a HTTP PUT request. Based on this data, it can generate a meaningful signature for the entire specific request with ports for input and output, which can be added as a part of some threat interface to the risk model.



http://192.168.128.128:8090/cce/web/#rtoverview:ALL/local/OSGI-CCE/local-OSGI-CCE/1

Initial Heap Size (MB)	Maximum Heap
256	512

PUT http://192.168.128.128:8090/cce/configuration/data/local/OSGI-CCE/COMMON-MEMORY

Fig. 3. The RACOMAT assistant for analyzing HTTP dynamically

A complete threat interface represents a component of the system, eventually in a specific state, and the immediately related risk analysis artefacts (i.e. weaknesses, attack patterns, faults …). In the case study presented here, there were already initial risk assessment artefacts which are imported from the ARIS tool into the RACOMAT tool. This includes a list of potential vulnerabilities for the entire web interface component, which is represented as an abstract threat interface in the RACOMAT risk model after the import. From the dynamic analysis, the RACOMAT tool generates more detailed threat interface descriptions for specific states. These state dependent threat interfaces can be linked with the more abstract threat interface for the entire Command Control web interface. The analysts can quickly go to through the list of potential vulnerabilities identified for the abstract threat interface and decide which of the vulnerabilities might exist for the more detailed state dependent threat interfaces and add those to the state dependent threat interfaces. This is basically like using a check list.

Furthermore, different state dependent threat interfaces can be linked with each other so that the sequence of how the states were reached is represented correctly in the risk model.

In addition to recording information about the signature of requests, the RACOMAT tool can of course also extract the value for each parameter in a request. This information can eventually be used to decide about what type of information might be expected, e.g. an ASCII string or an integer. This might already indicate which potential weaknesses should be investigated most carefully.

Even more important, the values themselves might become vital for any following security test case generation and test execution. First, the recorded values can be used again in order to reach another state in a sequence of states. Second, the same values can eventually also be used to test for vulnerability to reply attacks. Third, valid input values can be used very well as a starting point for fuzzing in order to generate slightly altered values as new test cases. These test cases altered values that are eventually only close to valid input values are good test candidates for analyzing the security of the system under test against manipulated input.

4.3 Automated Risk-Based Security Testing

After the semi-automated dynamical analysis, the risk model is much more detailed, but it still does not contain enough information to start security testing without manual effort. The RACOMAT tool provides assistants that can add more information from literature. For example, the CAPEC assistant can be used to add all the relevant related attack patterns from the Mitre CAPEC database for each identified potential vulnerability in a state dependent threat interface. Thereby, the attack patterns can be immediately linked with the related vulnerabilities and threat interface ports.

The attack patterns can be seen as security testing instructions. However, they do not contain executable test cases or machine readable test templates. The RACOMAT tool does provide an extendable library of predefined security test patterns which can be used to generate and execute test cases automatically once the pattern is instantiated. If no appropriate test patterns exist in the library, the tool allows its users to create new test patterns within the tool and to upload them to the library for sharing. Security test patterns are automatically associated with the attack patterns that can be tested using them. For instantiation, all that has to be done is assigning the potential observable results (i.e. unwanted incidents) to some output ports of the threat interfaces.

Based on likelihood and consequence estimates in the risk model, the RACOMAT tool can calculate the priority of the test patterns in order to spend the testing budget for the most critical tests. Likelihood estimates need only be made for base incidents. The RACOMAT tool can calculate likelihood values for dependent incidents by doing Monte Carlo simulations over the risk model. Of course, this requires that the relations between incidents are modeled accurately. Using the RACOMAT tool, dependencies between faults or incidents can be modeled in detail using directed weighted relations and gates. This might require some manual work. Without creating an accurate model, simulations or other calculations for dependent likelihood values are impossible.

Given an appropriately instantiated test pattern, test generation, test execution and test result aggregation are at least semi-automated. But for example for overflow tests, even complete automation is achievable using the RACOMAT tool. The entire security testing process is controlled from the risk assessment perspective, there is no gap in the workflow.

4.4 Test-Based Risk Assessment

Testing results can be used to identify new unwanted incidents that have not been considered in the risk model so far. The RACOMAT tool is capable of adding such unexpected incidents semi-automatically to the risk graphs.

Furthermore, test results should be used to create a model that approximates the behavior of the tested parts accurately so that this model can be used in future incidents simulations which are used to calculate dependent likelihood values. Security testing in RACOMAT tool means trying to trigger unwanted incidents. The model that has to be crated should tell how likely it is that the tested incident will ever be triggered. So a likelihood expression should be interpolated from the raw test results. Likelihood expressions for Incidents are exactly the base for the RACOMAT tool incident simulations.

But how can a likelihood expression be interpolated from testing results if testing was not nearby complete? Since sound interpretation is highly dependent on the tests themselves, security testing metrics which contain interpolation functions can be chosen from the RACOMAT tool predefined suggestions or created and applied manually. Test patterns should indicate which predefined security test metrics are most appropriate to analyze the testing process and the test results. Any function of a security testing metric will expect information about the executed test cases and about the results that are observed with the help of the observation strategies of the test pattern. Some metric's functions might need further information, for example about the test execution time or about the entire time spend on testing including the instantiation of the test pattern. Security test patterns contain information that helps assigning the input parameters of the suggested metrics and calling the metric functions correctly.

This bridging between a test pattern and a suggested testing metric can work automatically or at least semi-automatically.

Hence, it is possible to update the risk graphs automatically with more precise likelihood estimates interpolated from test results or with new faults based on unexpected test results.

After each iteration in the iterative RACOMAT process (i.e. after each security testing of some attack pattern), an incident simulation should be made to calculate updated likelihood values for dependent incidents before the most pressures threat scenarios which should be tested next are selected by the RACOMAT tool test prioritization algorithm.

4.5 Reusability, configurations and high level composition

Reusable risk artefacts are one important result of the RACOMAT risk assessment process. Typically lots of security testing will be done for individual components. The risk models created for the components can be reused wherever the component is used. After modeling dependencies between the models of the components, the individual components do not have to be analyzed and tested again. The RACOMAT tool is capable of calculating likelihood values and risk values for any configuration that is modeled with the help of incident simulations.

Within the RACOMAT tool, the risk models used for incident simulations are directed weighted graphs with gates, somehow like fault trees. However, in RACOMAT, graphs are not required to be trees – nor to be acyclic at all. This allows to model mutual dependencies and things like build in repairing features easily, but it also makes simulations more challenging. To enable deterministic incident simulations in cyclic graphs, the RACOMAT tool requires users to break the loop at some point for single rounds in the simulation.

4.6 Continuing Risk Management

Creating a more detailed, more complete risk model with more precise values and being able to assess any composition in any context is not the final goal of a risk assessment process. While refining is a necessary step and while with the level of automation the RACOMAT tool offers this step is manageable, it also has some drawbacks. Especially, it makes the risk model way more complex. The managers do eventually not want to see all the details all the time. A good condensed overview is much better fitting their needs, especially if they can go into the details of the analysis if they decide to take a closer look. Therefore, after going into most detailed low level risk assessment, it is necessary to make sure that more abstract, higher level results are produced, too, and that these are linked with the low level results appropriately.

For example, for Command Central, finally only the risk artefacts identified for the entire abstract web interface are regarded to be of interest for the management. The risk values for the abstract high level threat interface of the web interface are updated based on the security test results for the many state dependent threat interfaces. The RACOMAT tool automatically calculates these updates with its incident simulations.

Additionally, it provides high level dashboard views to support the further risk management.

Currently experiments are going on trying to allow for even higher abstraction using grouping and tagging for the risks based on information from literature und existing catalogues. First results seem promising especially for the tagging because it offers more flexibility.

In general the RACOMAT tool can be used as a stand-alone tool. It covers the entire process of combined test-based risk assessment (TBRA) and risk-based security testing (RBST) shown in Fig. 2. Nevertheless, it is also possible to use other possibly more specialized tools for some steps in that process. In the Case study presented here, the results of the risk assessment are integrated back into the models used by the ARIS framework tools. ARIS is established and widely used by the security experts and the developers responsible for Command Central. Therefore, the RACOMAT tool exports results in a JSON format that ARIS tools can import. However, the some results cannot be imported into ARIS models. In order to make them accessible even for managers and developers who do not want to use the RACOMAT tool, the development of a RACOMAT server has started that provides views with different levels of detail through a web interface.

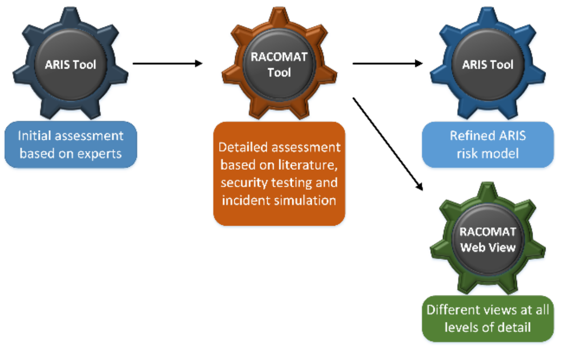

ARIS Tool
Initial assessment based on experts
RACOMAT Tool
Detailed assessment based on literature, security testing and incident simulation
ARIS Tool
Refined ARIS risk model
RACOMAT Web View
Different views at all levels of detail

Fig. 4. The different tools used within the case study

5 Conclusion and Future Work

The shown case study has lead massive extensions of the RACOMAT tool. Manly not to make the risk assessment theoretically more accurate, but to increase the level of automation and the usability was the major concern. With appropriate security test patterns and security testing metrics, it is possible to do a more or less completely automated Risk-Based Security Testing and Test-Based Security Risk Assessment with the RACOMAT tool.

The biggest problem at the moment is that there are only a few security testing metrics defined. The same is also true for the security test patterns – for many threat scenarios an attack patterns, there are currently no existing appropriate security test patterns. As long as these have to be created manually for each new attack pattern, the effort is just as high as for manual security testing. However, if there is a good pattern with sound metrics, then it can be instantiated for all occurrences of the same threat scenario with low manual configuration effort. Once there are extensive catalogues of patterns and metrics, then the concepts presented here will make the entire combined risk assessment and security testing process much easier and really safe a lot of manual work.

Creating a library of good security test patterns and security testing metrics is not a trivial task. With the RACOMAT tool, there is now at least a tool that supports creating and editing testing metrics as well as test patterns. This allows users to create the metrics and patterns needed.

Currently, a RACOMAT server for sharing test patterns, testing metrics and also for sharing reusable threat interfaces for entire components or programs is being developed. Sharing threat interfaces will be essential for allowing customers to do risk assessments for their specific environments, configurations and requirements themselves. For the future, the hope is that an open community will work with the threat interface, test pattern and testing metrics databases, developing them further in collaboration. Expecting that user feedback will be essential for quality assurance and for continuous improvement, user feedback will be taken serious and hopefully it will become a vital part of the open risk artefact, test pattern and testing metric databases.

References

1. M. S. Lund, B. Solhaug, K. Stølen: Model-Driven Risk Analysis – The CORAS Approach. Springer (2011)
2. International Standards Organization. ISO 31000:2009(E), Risk management – Principles and guidelines, 2009
3. International Standards Organization. ISO 29119 Software and system engineering - Software Testing-Part 1-4 , 2012
4. Bouti, A., Kadi, D.A.: A state-of-the-art review of FMEA/FMECA. International Journal of Reliability, Quality and Safety Engineering 1, 515–543 (1994)
5. International Electrotechnical Commission: IEC 61025 Fault Tree Analysis (FTA) (1990)
6. International Electrotechnical Commission: IEC 60300-3-9 Dependability management – Part 3: Application guide – Section 9: Risk analysis of technological systems – Event Tree Analysis (ETA) (1995)
7. Lund, M. S., Solhaug, B., Stølen, K.: Evolution in relation to risk and trust management. Computer 43(5), pp. 49–55, IEEE (2010)
8. Kaiser, B., Liggesmeyer, P., and Mäckel, O.: A new component concept for fault trees. In: 8th Australian workshop on Safety critical systems and software (SCS'03), pp. 37–46. Aus-tralian Computer Society (2003)
9. Papadoupoulos, Y., McDermid, J., Sasse, R., and Heiner, G.: Analysis and synthesis of the behaviour of complex programmable electronic systems in conditions of failure. Reliability Engineering and System Safety, 71(3), pp. 229–247, Elsevier (2001)
10. Viehmann, J.: Reusing risk analysis results - An extension for the CORAS risk analysis method. In: 4th International Conference on Information Privacy, Security, Risk and Trust (PASSAT'12), pp. 742-751. IEEE (2012), DOI: 10.1109/SocialCom-PASSAT.2012.91
11. Gleißner, W., Berger, T.: Auf nach Monte Carlo: Simulationsverfahren zur Risiko-Aggregation. RISKNEWS, 1: pp. 30–37. Wiley (2004), DOI: 10.1002/risk.200490005

12. Greenland, S., Sensitivity Analysis, Monte Carlo Risk Analysis, and Bayesian Uncertainty Assessment. Risk Analysis, 21: pp. 579–584. Wiley (2001)
13. Viehmann, J: Towards integration of compositional risk analysis and security testing using Monte Carlo simulation, presented at First International RISK Workshop 2013 Istanbul, published in: Bauer, Th., Großmann, J., Seehusen, F., Stølen, K., Wendland, M.-F. (Eds.): Risk Assessment and Risk-Driven Testing, pp. 109-119, Springer 2014, DOI: 10.1007/978-3-319-07076-6
14. Handbook: webMethods Command Central Help, Version 9.6, April 2014, Software AG Darmstadt Germany, http://documentation.softwareag.com/webmethods/wmsuites/wmsuite9-6/Command_Central_and_Platform_Manager/9-6_Command_Central_Help.pdf
15. Kloos, J., Hussain, T., and Eschbach, R.: Risk-based testing of safety-critical embedded systems driven by fault tree analysis. In: Software Testing, Verication and Validation Work-shops (ICSTW 2011), pp. 26–33. IEEE (2011)
16. Stallbaum, H., Metzger, A., Pohl, K.: An Automated Technique for Risk-based Test Case Generation and Prioritization. In: Proceedings of 3. Workshop on Automation of Software Test, AST'08, Germany, 2008, pp. 67-70.
17. Smith, B.: Security Test Patterns (2008). http://www.securitytestpatterns.org/doku.php
18. Erdogan, G., Seehusen, F., Stølen, K., Aagedal, J.: Assessing the usefulness of testing for validating the correctness of security risk models based on an industrial case study. Proc. Workshop on Quantitative Aspects in Security Assurance (QASA'12), Pisa (2012)
19. Benet, A. F.: A risk driven approach to testing medical device software. In: Advances in Systems Safety, pp. 157–168. Springer (2011)
20. Großmann, J; Schneider, M.; Viehmann, J.; Wendland, M.-F.: Combining Risk Analysis and Security Testing; ISoLA 2014 Corfu
21. Federal Office for Information Security (BSI): IT-Grundschutz Catalogues, Bonn Germany 2013, https://www.bsi.bund.de/EN/Topics/ITGrundschutz/ITGrundschutzCatalogues/itgrundschutzcatalogues_node.html
22. MITRE: Common Attack Pattern Enumeration and Classification, MITRE 2015, http://capec.mitre.org/
23. MITRE: Common Weakness Enumeration, MITRE 2015, http://cwe.mitre.org/data/index.html
24. MITRE: Common Vulnerabilities and Exposures, MITRE 2015, https://cve.mitre.org/cve/cve.html