Towards Data-driven Continuous Compliance Testing

Andreas Steffens
Research Group Software Construction
RWTH Aachen University
Aachen, Germany
steffens@swc.rwth-aachen.de

Horst Lichter
Research Group Software Construction
RWTH Aachen University
Aachen, Germany
lichter@swc.rwth-aachen.de

Marco Moscher
RWTH Aachen University
Aachen, Germany
marco.moscher@rwth-aachen.de

***Abstract*—Recent studies show that security vulnerabilities are caused by neglecting best-practices for the configuration of software and the underlying infrastructure. Due to the rising complexity of software systems and the accelerated speed of software releases using mechanisms like continuous delivery the problem gets even more challenging. Existing processes and methods are not adequate to cope with these challenges. This paper proposes an approach for continuous compliance testing. Using well-known methods from software testing, this approach enables an organization to define, organize, and execute compliance tests in a structured and reusable way. We focus in our approach onto integrating a software-centric point of view for modeling compliance requirements. By embedding our approach into a deployment pipeline automated continuous compliance testing can be realized.**

***Index Terms*—software testing, system testing, continuous software engineering, compliance, security, devops**

I. Introduction

Software must run on a defined operating infrastructure to fulfill its purpose. In recent years, the assessment of infrastructure gained more and more attention from a security perspective. OWASP identified the *Misconfiguration of Server* as one of the top10 security risks [1] for web applications. The misuse of the underlying infrastructures by apps even rose to the first place for mobile platforms.

Therefore, Organizations like the Center of Internet Security (CIS)[1], NIST[2] and the German BSI[3] developed and published tools, guidelines, checklists and benchmarks, which offer the opportunity to assess a system.

To assess the operating infrastructure, so called compliance tests need to be performed. Lewis and Bassetti [2] define compliance testing as a general practice to determine if all mandatory requirements from a specification are met. In our context we use the more focused defintion. The ISTQB[4] [3], [4] and the ISO Standard 27002 [5] for cyber security defines compliance as the adherence of legal and contractual regulations and further external requirements like results from audits and reviews. In general compliance tests are based on external compliance rules, policies and standards.

[1] www.cisecurity.org
[2] www.nist.gov
[3] www.bsi.de
[4] International Software Testing Qualifications Board

This paper is motivated by a lack of a structured and reliable way to organize and assess compliance aspects, which is intensified due to increasing software and infrastructure complexity. Beside this, the challenges of growing pace in software development through agile and automated delivery of new functionality and software, are inevitable. Concepts and methods like continuous delivery [6] or DevOps [7] offer solutions for these problems. Compliance testing is forced to keep up with these circumstances.

Otherwise, the compliance status of the operating infrastructure will erode, which leads to system misconfiguration and security vulnerabilities. In general a system misconfiguration has serious impact on the system and software security as stated in the OWASP Top 10 report. Thus the automation of compliance testing is necessary.

Besides testing the operating infrastructure for global standards and regulations, the operating infrastructure has to be compliant to the software and its configuration. Therefore, from a software perspective new compliance requirements are introduced. Any software vendor should be interested, that their software products and their customers infrastructure are compliant. For example, if a software component requires a special cryptographic library to provide sufficient security, it is necessary to check if this requirement is satisfied before bringing the software into operation. As a consequence, the infrastructure provided by the customer needs to be tested for compliance. Therefore, the vendor should be able to define custom compliance requirements for single or multiple related software components. Consequently, these specific software compliance requirements need to be tested as well, before deployment, after a deployment and during operation. Basically, compliance needs to be ensured continuously..

Although, it is easy to define certain compliance requirements on small scale, it gets more difficult when enlarging the scope towards complex enterprise systems. Hence a modular approach of defining, assigning, and checking those compliance requirements as already established for various standards from a *software-centric perspective* is needed.

This paper introduces an novel approach to fill this gap. The main contribution is an unified model for specifying, managing and executing compliance tests with a software-centric viewpoint. It supports the stakeholders to ensure, that

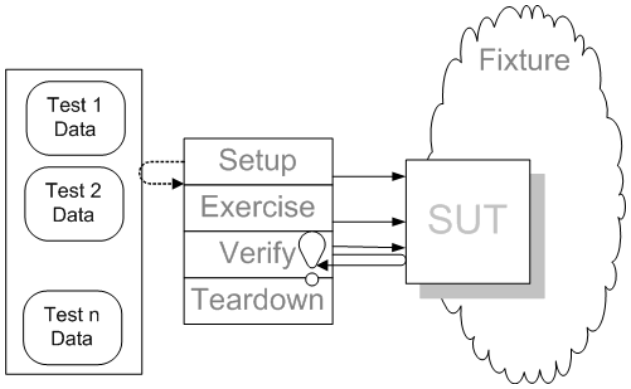

Fig. 1: Data-Driven Testing [8]

the unique landscapes of software systems are compliant. In addition it enforces the integration into proven concepts like Continuous Integration and DevOps.

The paper is structured as follows. The next section will introduce the needed background of continuous delivery and data-driven testing. In section 3 will define the concept of a software-centric point of view for compliance testing. Based on theses results we introduce our model for continuous compliance testing. The subsequent section will deal with some important aspects as reusability and validation and the integration into DevOps. COMET, a tool implementing our approach, will be briefly described in the next section, followed by a short discussion about related work and existing tools for compliance testing. A final section will provide the conclusion and some proposals for future research in this area.

II. Background

A. Data-Driven Testing

Data-Driven Testing (DDT) [8] is a testing technique, where one test may contain more than one set of data values. Because the same test logic is applied in different test cases. Based on parametrized tests, data is injected into the tests from an external data source before executing the actual tests. Figure 1 illustrates the process of DDT. In the setup phase, the external data is loaded into the test framework and for each data set a dedicated test is executed on the system under test (SUT). DDT increases the reuse of test code and eases the maintenance of this code base. In the approach presented in this following we derive compliance test data from a model comprising rules, software components, and parametrized tests. All of these can be specified by different stakeholders.

B. Continuous Delivery

Humble and Farley [6] define continuous delivery as a set of practices, which enables to speed-up, automate and optimize the delivery of software artifacts to the customer with even higher quality in a continuous manner. These practices are highly related to common agile practices like iterative and incremental development. Scrum [9], an agile project management method, requires to produce a shippable increment at the end of each sprint.

Continuous delivery uses an automated development infrastructure, called deployment pipeline, which automates nearly every step of the delivery process. Each commit of a developer enters the deployment pipeline and an automated process is started, which produces a new software increment as a result artifact.

The deployment pipeline incorporates all activities known from continuous integration [6] as automatic build, unit testing, and static code analysis. In addition to these, the pipeline performs testing activities like integration, performance, and security testing. All these tasks are executed in a defined order of stages. After each stage the test results a evaluated at a quality gate, which stops the processing if the quality conditions are not met. If all quality gates are passed, the software artifact is stored and can be accessed and used from external clients; it is released.

If the software is directly used in an production environment this extension is called continuous deployment. Continuous delivery favors the principle of shift left [10], which means to test as much as early as possible. This also includes to run the tests in an environment which follows the properties of the target environment, e.g. the production system. To achieve this, the specific properties of the target environment needs to be known and to be ensured in every stage of the deployment pipeline, the environment needs to be compliant.

Compliance testing and continuous delivery are compatible to extend each others functionality and usefulness. The following section will introduce an approach to specify these compliance requirements for environments. Finally, we present a tool to run compliance tests using data-driven testing in a continuously manner.

III. Compliance Testing from a Software-centric Point of View

The existing approaches to compliance testing like the CIS Benchmarks or the guidelines published by BSI target low level and minimal requirements for secure infrastructures. For companies and the affected stakeholders like the Chief Security Office (CSO) or the Chief Information Officer, it is hard to apply these guidelines to their current set of software systems. In general, most organizations apply the basic levels. We argue, that this level of compliance testing needs to be improved by extending the viewpoint from the infrastructure to the software systems and complete landscapes running on these infrastructures. We propose to shift to a software-centric perspective. Each software introduces additional individual requirements for the configuration of infrastructure and even of related software. These requirements need to be considered when conducting a compliance test. The following example explains the purpose and the need for a software-centric point of view. Consider a software system with the following requirements regarding the web server software Apache httpd[5]

[5]http://httpd.apache.org/

running on a Linux server node:

- The software Apache httpd runs as the user www-data.
- Apache httpd uses the local TCP port 80 to provide the web pages to external clients via HTTP.
- The served web pages are stored in a directory which can be specified with the property DOCUMENT_ROOT.
- This directory needs to be readable and writable.
- It writes its logs to the directory /var/log/apache2.

From these requirements we can deduce new configuration related requirements which need to be fulfilled to be able to run the Apache httpd software. Failing one of these requirements will result in a non-operational system. So, these requirements add new rules, to ensure that the system is configured correctly and compliant to the software.

Software systems nowadays do not consist of one software component but of multiple. Complex software applications form a distributed landscape of software components running on different nodes in the network. These landscapes can be specified and compiled dynamically and will change during the application lifecycle.

Consequently, from a software point of view all the configuration requirements of the different software components need to be considered when performing a compliance test for this software system.

Based on these insights, the following sections introduce a well-defined domain model for compliance testing which incorporates a software-centric point of view.

IV. Continuous Compliance Model

The proposed Continuous Compliance Model can be seen as a domain model for continuous compliance testing. Following the principles of Domain-Driven Design [11], the model is divided into five bounded contexts which represents specific concerns regarding compliance testing.

Figure 2 visualizes the continuous compliance. The contexts Operation and Software represent the software-centric point of view. Compliance inhabits concepts regarding compliance rules and their organization. The Technical context deals with the implementation of assessments as parameterized tests. The Execution context contains concepts necessary to perform, evaluate, and save tests and their results. The following sections will elaborate on the identified concepts and their relationships.

A. Compliance

A *Compliance Rule* defines the shape of a specific aspect or property of the SUT which needs to be checked. If the rule is fulfilled the system is compliant regarding this specific rule. A compliance rule can comprise multiple varying *Compliance Tests* to assess the desired property.

Examples of compliance rules for a secure web service could be:

1) The SUT provides a service on the local port TCP 443.
2) The SUT uses the protocol HTTP to communicate on port 443.
3) The SUT offers encrypted communication secured by SSL/TLS on Port 443 with a minimal version of TLS 1.2.
4) The SUT uses a valid SSL certificate to encrypt its communication on port 443.

A *Compliance Rule Set* is a collection of specific compliance rules which belong together and need to be checked together. The examples from above can be aggregated to a rule set for checking a web server.

As compliance rules can be very specific, this leads to a reduced reuse potential of this rules. Therefore, we introduce variables which enable and support reuse of rules. A compliance rule becomes parameterized in the same way as parametrized tests [8]. A more generalized version of the rules from above could look as following:

1) The SUT provides a service on the local port TCP VAR_PORT.
2) The SUT uses the protocol HTTP to communicate on port VAR_PORT.
3) The SUT offers encrypted communication secured by SSL/TLS on port VAR_PORT with a minimal version of TLS 1.2.
4) The SUT uses a valid SSL certificate to encrypt its communication on port VAR_PORT.

The variable VAR_PORT is specific to the current SUT and can vary for different systems and/or different software components. The version of TLS could be also transformed into a variable, but a compliance rule should be defined as precise as possible (e.g., we do not want to allow to change the TLS version to on older less secure one). Organization wide policies can be applied at this level, the need for more context specific data (as a port number) can be delegated to higher level software components.

The relation between compliance rules and software components will considered closer in a subsequent section.

B. Compliance Tests

To check for compliance we need an automated way to evaluate compliance rules and apply them to actual systems and environments. Our approach to compliance testing differentiates between compliance rules and the way they are evaluated. This is represented by the concept of a *Compliance Test*. A compliance test is an implementation of an assessment of system properties. A compliance test can be executed and evaluated automatically. It offers a defined set of variables, which needs to be assigned before execution. The final test code will be derived and generated during execution of the test.

The technology used to evaluate the required properties for a compliance rule, can be chosen freely. Hence, the technology is not important and our approach is therefore technology agnostic.

Listing 1: Parameterized Compliance Test with InSpec

```
control "TestForOpenPort" do
  impact 1.0
```

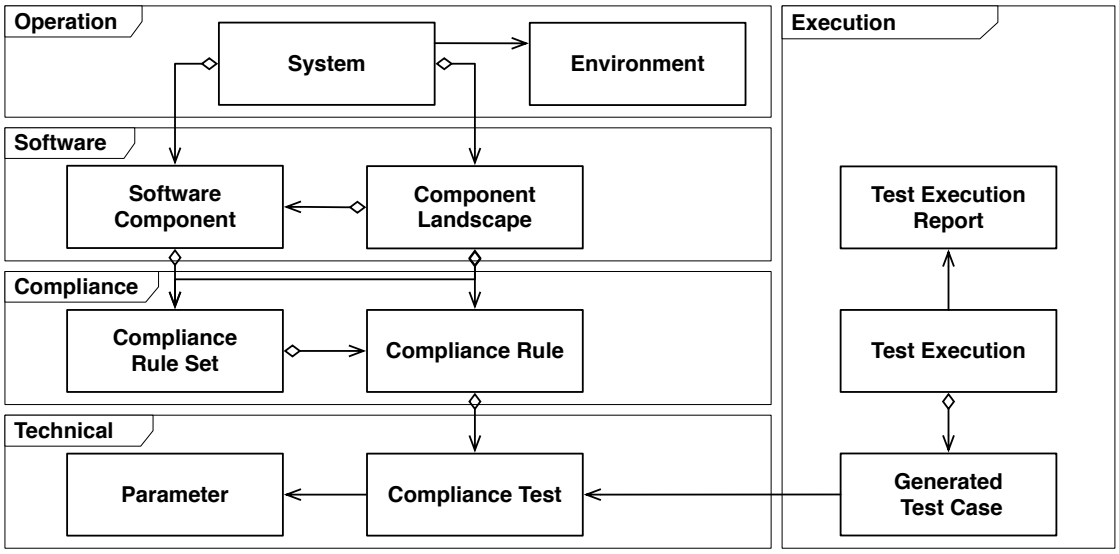

Fig. 2: Continuous Compliance Model

```
  describe port(VAR_PORT) do
      it { should be_listening }
  end
end
```

Listing 1 defines a compliance test using InSpec[6] to check if a certain port is used.

A compliance test can be used by a compliance rule to asses for its required system properties. A compliance rule can use multiple compliance tests. As the implementation of compliance test is unrelated to specific compliance rules, the development of compliance tests can focus on offering a highly reusable set of tests as a library for specifying compliance rules.

C. Software Components and Landscapes

As there is a need to incorporate a software-centric point of view into compliance testing, we introduced Software Component, Component Landscape and System as model elements into the Continuous Compliance Model.

A Software Component represents a software artifact/product that has requirements regarding the configuration of the infrastructure and defines desired properties and aspects of a system or node. A software component can define variables to model properties which are required but can be assigned to context-specific values.

As an example of such a component conside Apache httpd. A corresponding variable could be the PORT for the web server software with a default value (e.g., PORT = 80).

A *Component Landscape* is a set of software components which form a working application or product. A component landscape can be distributed among different nodes in a network.

[6]https://www.chef.io/solutions/compliance/

Software components and component landscapes can be associated with compliance rules and rules sets to model specific compliance requirements.

D. Software System

A Software System is the representation of a real operating software system running on various nodes in a network.

A software system is a specific set of component landscapes and components and comprises an Environment, which specifies the actual nodes in a network, where the components are running. Software components and landscapes can be part of multiple systems. On such a system a compliance test can be conducted, as it is accessible and assessable by compliance tests.

Based on these concepts, the necessary compliance tests to assess a system can be derived. Figure 3 depicts a concrete compliance testing scenario for the Apache httpd web server and a custom software component. These two components form a component landscape which is instantiated as a software system running on AWS. For Apache httpd the specified compliance rules and tests are added to the model.

E. Continuous Compliance Testing

To perform compliance testing in a continuous manner and to embed it into a continuous delivery process, we have to define possible interaction points, where a deployment pipeline can integrate and execute a compliance model. The software or framework operating on the continuous compliance model needs to manage multiple test executions to produce reports which can be evaluated by the pipeline.

To generate executable test cases, the compliance model which is represented as directed acyclic graph (DAG) is traversed. Each path represents an individual test case, which will be executed and evaluated. While traversing the model,

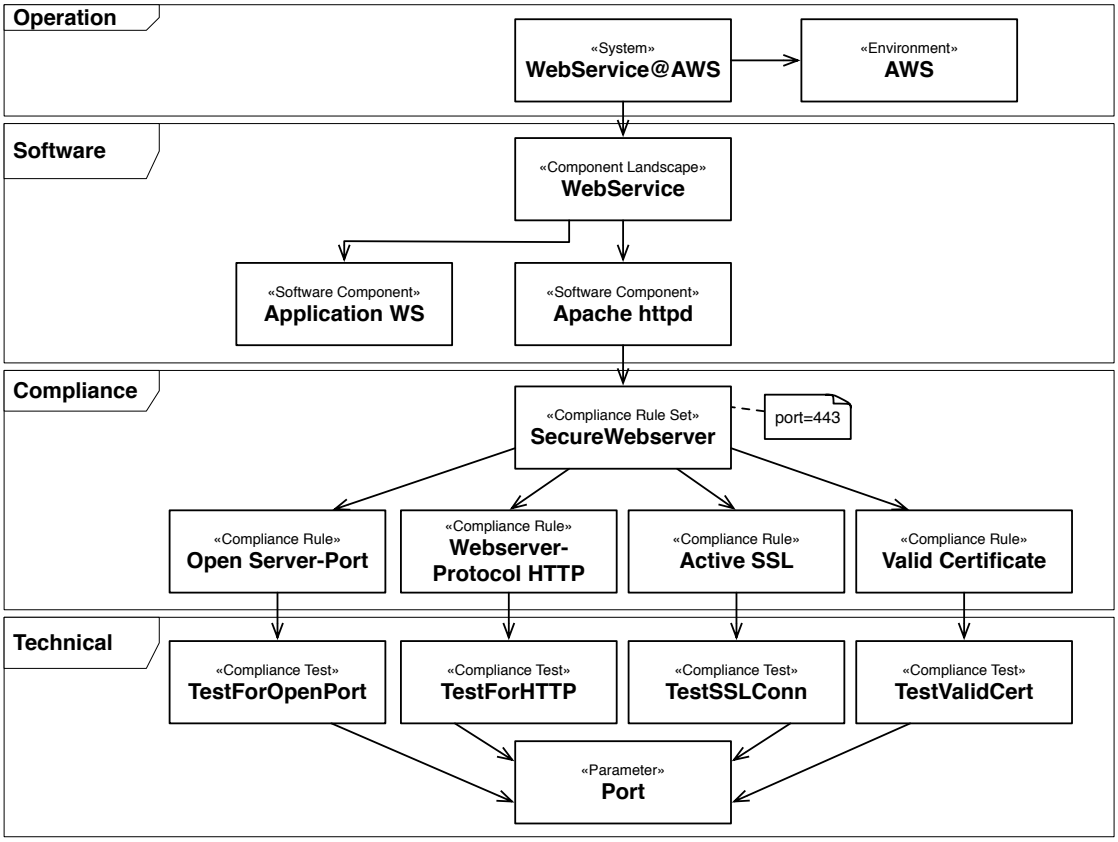

Fig. 3: Compliance Model for Apache httpd

variables are resolved top-down to their final corresponding value. At the technical level, all variable values are injected as parameters for the compliance tests, which are implemented as parameterized tests. So each path of the graph results in a Generated Test Case.

A *Test Execution Report* is an artifact containing all test results from all *Test Executions*, which ran a specific instance of the *Continuous Compliance Model*. To ensure traceability a test execution includes the executed generated test cases.

The example compliance model depicted in Figure 3 leads to the generation of four different test cases. The variable PORT defined on the software level will be delegated down to the technical level and injected as a parameter for each test.

1) Validation & Simulation: Based on a complete model validation and simulation activities can be performed to gain insights about the current compliance status without even running the tests explicitly, e.g. missing or conflicting variable definitions can be discovered on model level by analyzing different paths and the potential test cases. Contradictory rules can be discovered when modeling a new software landscape or system. Techniques like constraint solving can be applied to find a compliant configuration for the modeled system.

2) Risk computation: To each compliance rule a risk value can be assigned, which indicates the potential risk if it is not fulfilled by the SUT. All risk values will be aggregated along the paths and added in the root node, the system element, to gain a singular risk value for this specific software in this specific environment at the time of execution of the compliance tests.

3) Reusability: Our approach supports and enforces reuse of technical, compliance, and software model elements when modeling and adding new elements. Current compliance standards like CIS or BSI can be easily modeled as compliance rules or compliance rule sets and then be assigned to certain software components which are operated in the organization. By means of variables, the rules can be tailored to specific environments and systems without touching the implemented standard itself. Test case implementations can be reused in different rules for different purposes. In the end the tests, rules, and components form a standardized organization-wide library of compliance related artifacts.

4) Continuous Delivery: Continuous delivery and its technical implementation, the deployment pipeline, define various stages where testing activities can be performed. As continuous compliance testing is a form of nonfunctional testing [2] it can be performed in a similar fashion as performance tests or security tests.

But in addition the compliance rules and tests can be used for checking test systems and environments used during a pipeline run, to ensure a proper compliance state before per-

forming additional functional testing or to test deployment and provisioning mechanisms. Chef Automate[7] uses this approach to test Infrastructure As Code [12].

5) DevOps: As the compliance model integrates information about the desired target production environment and about how the software is operated it also supports the DevOps mindset [13]. Operational information is managed by model-based and generative software engineering methods.

By supporting and implementing ShiftLeft, developers get in touch with operational principles and constraints. In agile projects this will have an impact on software design and architecture.

In addition, former non-related activities like IT-Governance will be aligned with software development. Due to the decoupled structure, the compliance model can be defined by different stakeholders. Compliance rules may be defined by a CSO but the corresponding compliance tests are provided by developers. We skip the organizational and cultural issues of compliance modeling and testing as this is out of the scope of this paper.

V. Tooling: COMET

COMET is a web-based continuous compliance testing framework, based on the continuous compliance model (see figure 2) The user of COMET can add compliance tests, rules and software components to build a continuous compliance testing scenario. The architecture of COMET follows the bounded contexts introduced in Section IV defining technical, compliance, software, operation and execution related areas. The user can define and modify entities of these areas independently. To build a complete compliance testing model the user needs to provide or reuse elements of all areas.

COMET is able to execute a continuous compliance model, by traversing the model, resolving the variables with the system, software, or rule specific input-data, injecting these data into the parametrized compliance tests, then generates and executes the generated test cases. In the end the software will provide a report with the results of the execution.

Figure 4 depicts the COMET report screen after the execution. Sensitive informations are defaced by blurring these parts in the screenshot. All results, failed and successful tests, can be analyzed by the user. The system reports the accumulated risk value derived from all failed compliance tests. Currently, COMET supports only InSpec as possible test technologies, but the architecture provides a defined hot-spot to add new testing technologies.

VI. Related Work

The area of compliance and compliance testing has been in the focus of research and industry for quite some time. Frameworks like COBIT [14] and ITIL [15] are reference models for IT-Governance which include also activities regarding compliance. These models operate on a higher more process-oriented level and rely on documents and manual audits, e.g. ITIL defines a compliance registry where all compliance requirements and the intended way they should be assessed are documented. But it offers no mechanism to automate these assessments.

From a technical perspective some tools has been developed to perform compliance tests on a running system.

But, the research topic compliance as code and continuous compliance testing are relatively new and not yet that present. Tools like Chef InSpec[8], RedHat OpenScap[9], UpGuard[10], CIS-CAT[11] and others allow to define and execute compliance tests. Although, the possibilities to automate compliance testing, to integrate it into continuous delivery and to manage multiple complex environments are limited. A software-centric point of view is incorporated in any of these tools. But, COMET builds upon these technologies and uses to implement compliance rules.

Academia and industry are discussing about extending the scope from DevOps to SecDevOps, DevSecOps or DevOpsSec [16], [17] and all emphasize that security should be embedded into the existing approaches of DevOps. To this extend, first attempts facing automation and integration into DevOps exist and enable security testing and compliance verification in an automated manner. Hasicorp's Sentinel[12] defines policy-as-code, equivalent to a compliance rule, which can be integrated into organization's deployment and cloud products. But, modeling and maintaining complex compliance scenarios with this approach is quite hard.

VII. Conclusion and Future Work

Compliance testing is getting more and more attention as for example misconfigured infrastructures are one of the top security risks for software systems. A lot of effort has been spent by organizations like NIST or the BSI to collect knowledge and best practices to come up with guidelines and benchmarks. But these are quite general and focus on the infrastructure level. So, an organization operating or developing software struggles with ensuring compliance and at the same time incorporating all the necessities of their various software products.

This paper presents a novel approach to continuous compliance testing, which enables and provides a structured way to specify, organize and execute compliance tests. The approach focuses on integrating a software-centric view into compliance modeling. Therefore, it becomes necessary to model the actual software components in addition to the infrastructure and their relation to compliance rules. Compliance tests are derived by analyzing the specified software elements and their assigned compliance rules. The tailoring of these tests to specific environments is supported by using concepts of data-driven testing with the extension that the data is generated on demand based on the compliance model during execution. This eases the

[7] chef.io

[8] https://www.chef.io/solutions/compliance/

[9] https://www.open-scap.org/

[10] https://www.upguard.com/

[11] https://learn.cisecurity.org/cis-cat-landing-page

[12] https://docs.hashicorp.com/sentinel/

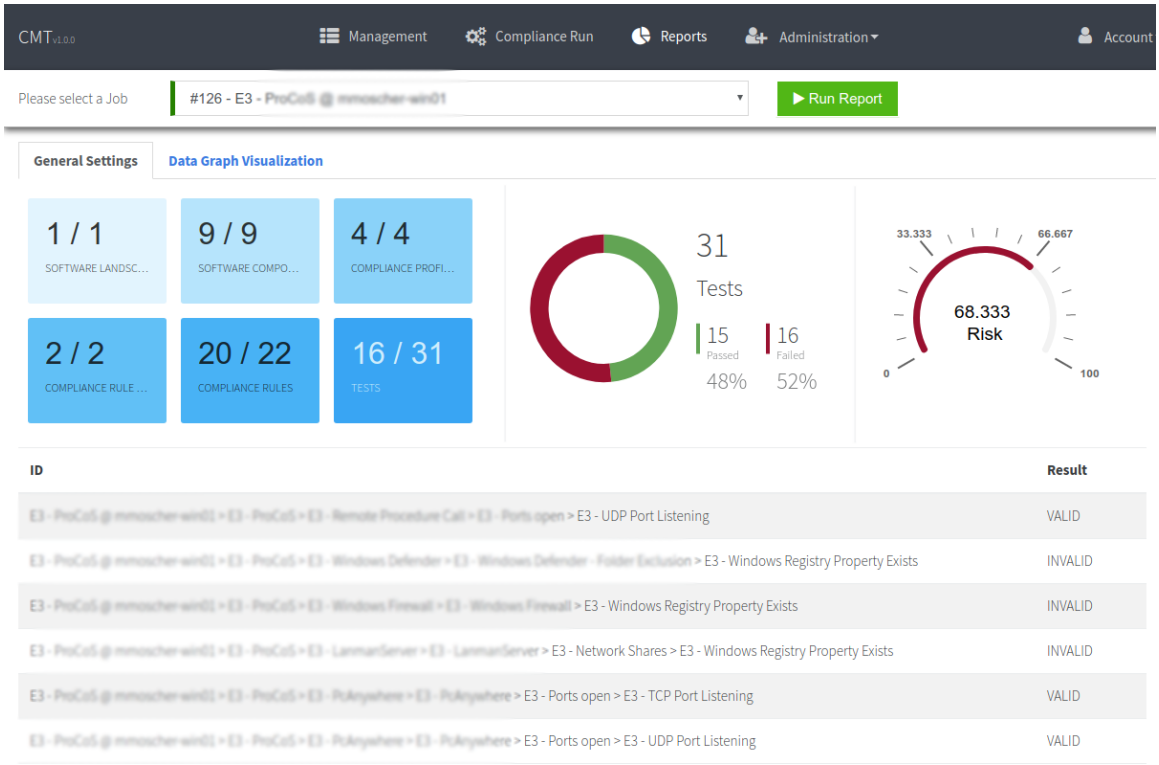

Fig. 4: Report Screen of COMET

reuse of compliance rules as these are similar to parameterized tests.

Due to its generative nature, the approach can be easily automated and embedded into a continuous delivery process and so is paving the way to continuous compliance testing. Applying compliance testing in early stages of the deployment pipeline, by using the rules in integration test environments, supports fast and early feedback of the compatibility of the target environment with the most recent software versions.

We offered a first glimpse on our tool COMET, which implements our approach. A conducted case study, which uses COMET in an industrial context is out of scope of this paper.

In future we plan to further evaluate our approach and gain more insights into the challenges of compliance testing. Especially the integration of different stakeholders like developers, security experts, product owners and managers into the process of developing and maintaining continuous compliance models is challenging. We seek to integrate concepts of business IT alignment to tackle this challenge.

References

[1] OWASP Foundation, "OWASP - Top 10 2013-A5-Security Misconfiguration," 2013. [Online]. Available: https://www.owasp.org/

[2] W. E. Lewis and W. H. C. Bassetti, *Software Testing and Continuous Quality Improvement, Second Edition*. Boston, MA, USA: Auerbach Publications, 2004.

[3] International Software Testing Qualifications Board, "ISTQB Standard glossary of terms used in Software Testing." [Online]. Available: http://glossar.german-testing-board.info/#eng

[4] D. Graham, E. V. Veenendaal, I. Evans, and R. Black, *Foundations of Software Testing: ISTQB Certification*. Intl Thomson Business Pr, 2008.

[5] D. I. für Normung Normenausschuss Informationstechnik und Anwendungen, *E DIN ISO/IEC 27002: Leitfaden für das Informationssicherheits-Management (ISO/IEC 27002:2014)*, ser. DIN-Normen: Deutsches Institut für Normung. Beuth, 2014.

[6] J. Humble and D. Farley, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*, 2010.

[7] L. Bass, I. Weber, and L. Zhu, *DevOps: A Software Architect's Perspective*, 1st ed. Addison-Wesley Professional, 2015.

[8] G. Meszaros, *xUnit test patterns: Refactoring test code*. Pearson Education, 2007.

[9] K. Schwaber and M. Beedle, *Agile Software Development with Scrum*, 1st ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2001.

[10] Donald Firesmith, "Four types of shift left testing," 2015. [Online]. Available: https://insights.sei.cmu.edu/sei_blog/2015/03/four-types-of-shift-left-testing.html

[11] E. Evans, "Domain-Driven Design: Tackling Complexity in the Heart of Software," *Folia primatologica international journal of primatology*, vol. 70, no. 5, p. 560, 2003.

[12] K. Morris, *Infrastructure as Code: Managing Servers in the Cloud*. O'Reilly Media, 2016.

[13] L. Bass, I. Weber, and L. Zhu, *DevOps: A Software Architect's Perspective*, 1st ed. Addison-Wesley Professional, 2015.

[14] I. S. Audit and C. Association", *Cobit 5*. ISA, 2012.

[15] W. Johannsen and M. Goeken, *Referenzmodelle fr IT-Governance - strategische Effektivitt und Effizienz mit COBIT, ITIL und Co: mit einem Praxisbericht*. dpunkt.verlag, 2007.

[16] J. Bird, *DevOpsSec: securing software through continuous delivery*. O'Reilly Media, 2016.

[17] V. Mohan and L. B. Othmane, "Secdevops: Is it a marketing buzzword? - mapping research on security in devops," in *2016 11th International Conference on Availability, Reliability and Security (ARES)*, Aug 2016, pp. 542–547.