RQCODE – Towards Object-Oriented Requirements in the Software Security Domain

Ildar Nigmatullin, Andrey Sadovykh, Nan Messe, Sophie Ebersold, Jean-Michel Bruel

RQCODE – Towards Object-Oriented Requirements in the Software Security Domain

Ildar Nigmatullin
Innopolis University and University of Toulouse
Innopolis, Russia and Toulouse, France
i.nigmatullin@innopolis.ru

Andrey Sadovykh
Innopolis University and Softeam
Innopolis, Russia and Paris, France
a.sadovykh@innopolis.ru

Nan Messe
University of Toulouse – IRIT – CNRS
Toulouse, France
nan.messe@irit.fr

Sophie Ebersold
University of Toulouse – IRIT – CNRS
Toulouse, France
sophie.ebersold@irit.fr

Jean-Michel Bruel
University of Toulouse – IRIT – CNRS
Toulouse, France
jean-michel.bruel@irit.fr

***Abstract*—For the last 20 years, the number of vulnerabilities has increased near 20 times, according to NIST statistics. Vulnerabilities expose companies to risks that may seriously threaten their operations. Therefore, for a long time, it has been suggested to apply security engineering – the process of accumulating multiple techniques and practices to ensure a sufficient level of security and to prevent vulnerabilities in the early stages of software development, including establishing security requirements and proper security testing. The informal nature of security requirements makes it uneasy to maintain system security, eliminate redundancy and trace requirements down to verification artifacts such as test cases. To deal with this problem, Seamless Object-Oriented Requirements (SOORs) promote incorporating formal requirements representations and verification means together into requirements classes.**

This article is a position paper that discusses opportunities to implement the Requirements as Code (RQCODE) concepts, SOORs in Java, applied to the Software Security domain. We argue that this concept has an elegance and the potential to raise the attention of developers since it combines a lightweight formalization of requirements through security tests with seamless integration with off-the-shelf development environments, including modern Continuous Integration/Delivery platforms. The benefits of this approach are yet to be demonstrated in further studies in the VeriDevOps project.

***Index Terms*—Requirements Engineering, Software Security, Security Testing, STIG, OO Requirements, DevSecOps**

I. Introduction

Software systems are becoming increasingly complex over time, making quality assurance more difficult [1]. To ensure software quality that meets a sufficient level of customer expectations, quality characteristics must be specified, taking into account the intended use of a software product and system. Relevant quality characteristics have been proposed in many quality models, including ISO standards, to evaluate software products properly. However, the number of threats increases, and the attacks are more and more various. For example, they exploit third-party vulnerabilities or profit from the negligence of system administrators when setting hard-coded credentials [2].

This paper considers different approaches and practices that integrate the security aspect into the lifecycle of software products, such as security testing, DevSecOps, and security requirements management. Firstly, security testing considered in this paper relates to the security quality properties as defined in ISO 25010 standard [3]. Security testing identifies whether the specified or intended security properties are correctly implemented [4]. Secondly, we aim at addressing the security aspect throughout the software's lifecycle in a continuous manner. This may relate to the DevSecOps approach, which focuses on the continuous cycle of addressing new coming security requirements and threats. Finally, we consider security requirements management. More precisely, we study how to make security requirements traceable, testable, and maintainable. In the current state, developers are unable to carry out rapid security requirements assessments, especially in the context of DevSecOps [5].

II. Software Security

ISO standard determines the following security properties as quality attributes [3]:

- Integrity: the property that a system, product or component prevents unauthorized access to, or modification of, computer programs or data.

- Non-repudiation: the property that actions or events can be proven to have taken place so that the events or actions cannot be repudiated later.

- Accountability: the property of being able to trace activities on a system to individuals, who may then be held responsible for their actions.

- Authenticity: the property that the identity of a subject or resource can be proved to be the one claimed.

The primary aspect to ensure security as a quality attribute is to consider it throughout the Software Development Life-Cycle (SDLC). Regarding SDLC, security aspects can be analyzed and taken into account in different stages, such as System Analysis, System Design, Coding, and Testing. Security is no less important in continuous integration and

continuous development. That is why the DevSecOps approach ensures security at every stage of the SDLC and helps verify every commit executed by development teams. Continuous delivery pipelines are meant to be a paradigm for contributing to the verification of each commit executed by developers. Therefore, the integration of automated security checks into the pipeline allows it to receive warnings at early stages and continuously monitor undiscovered security vulnerabilities at previous stages. The integrated approach to continuous security, scales in consistency with the growth of the business.

The security aspect is suggested to be considered as early as possible in SDLC to prevent potential damages caused by undiscovered vulnerabilities [6]. That is why our focus in this paper is on security requirements. Security requirements can be positive and functional, explicitly defining the expected security functionality of a security mechanism; or negative and non-functional, specifying what the application should not do [4]. The way of presenting security requirements impacts their verification. The more requirements are accumulated (e.g., received by change requests or appeared as ideas), the more resources are required to ensure their verification, including the completeness of requirements, their traceability and consistency from our point of view.

III. SOOR AND RQCODE

In this paper, we suggest using the object-oriented (OO) approach to make security requirements verification feasible. The OO paradigm fits large and complex programs that are actively updated and maintained. It provides features for easily defining complex objects by composing or augmenting the behavior of existing objects. These features significantly improve the reuse of an object by easily allowing it to be extended beyond its initial applications [7].

In this context, we promote that security requirements can be presented as an OO class template. The main role of a Requirement as Code (RQCODE) paradigm is to encapsulate the verification complexity and make the requirements template applicable and reusable across multiple systems. What remains to the specifier is to find what most closely formalizes the behavioral pattern implied by the target requirement and then inherits from it, providing system-specific details through the OO genericity and abstraction [8].

In Seamless Object-Oriented Requirements (SOOR) [9] requirements are encapsulated as object-oriented entities. This helps to decrease the complexity of verification and validation of requirements, while making requirements applicable and reusable across multiple systems. In SOORs, requirements are written as classes. A SOOR takes the form of a class that inherits from a SOOR template (SOORT), providing system specific details through the object-oriented (OO) genericity and abstraction. The example of SOORT in Java will be provided below, we call this implementation RQCODE.

The initial idea of RQCODE was born in an attempt to simplify the original SOOR concept by proposing an implementation in a widely-used programming language – Java. Moreover, we intend to offer formalization that is commonly understood by the software engineers by embedding test cases as enforcement, verification and validation means. We choose Software Security as an application domain, since it provides a vast area for experiments with a great variety of existing standard requirements and verification methods to be applied. Those security requirements are largely reused from project to project, which make us believe that the verification methods can be reused too.

The advantage of SOOR is that it can provide requirements representation in various forms, starting from a less-formal representation. In addition, the requirements object can encapsulate more formal representation in various formal notations or a set of test cases with complex behavior. In RQCODE, we concentrate on requirement representation in the form of a test case. We argue that this form is more convenient for developers, while it is sufficiently formal. Applying the SOORs allows us to focus on verification- and traceability-oriented specifications that are powerful enough to make the identified requirements practically reusable, and without duplicating.

To illustrate the SOOR and RQCODE concepts and how our approach ensures Security requirements verification, let us consider the following example of a security requirement from Windows 10 Security Technical Implementation Guide (STIG)[1]. STIG is meant as "a tool to improve the security of Department of Defense (DoD) information systems". The one for Windows ten is to be used in conjunction with other applicable STIGs, such as ones for Browsers, Antivirus, and other desktop applications. STIGs describe requirements, countermeasures and verification means designed to assist Security Managers, Information Assurance Managers, and System Administrators with configuring and maintaining security controls[2].

We will consider the following Window 10 STIG rule :

- V-63483: The system must be configured to audit Privilege Use – Sensitive Privilege Use failures. It maintains an audit trail of system activity logs that can help to identify configuration errors, troubleshoot service disruptions, analyze compromises, and detect attacks. The path to configure this security requirement is the following: *Configure the policy value for Computer Configuration Windows Settings - Security Settings - Advanced Audit Policy Configuration - System Audit Policies - Privilege Use - "Audit Sensitive Privilege Use" with "Failure" selected*.

This example demonstrates the behavior that is implemented with the Windows commands. The class hierarchy is based on configuration path for each requirement. We can apply Java class inheritance paradigm to represent the above requirements in the following object-oriented way:

```
Requirement ->
    CheckableEnforceableRequirement ->
    AuditPolicyRequirement ->
```

[1]https://www.stigviewer.com/stig/windows_10/

[2]https://ncp.nist.gov/checklist/629

```
    PrivilegeUseRequirement ->
    SensitivePrivilegeUseRequirement
```

In Java language syntax to present SOOR the requirements may look as follows:

```
abstract class Requirement
```

This class includes general attributes for storing requirements in natural language and output them in various representations such as plain-text or HTML [3]. Checking and enforcing are left as abstract methods to be specified further by class children. This class below contains the core logic for checking and enforcing methods.

```
public abstract class
    CheckableEnforceableRequirement extends
    Requirement
```

Here we check Audit policy requirements:

```
public abstract class AuditPolicyRequirement
    extends CheckableEnforceableRequirement
```

In these classes we define the common behavior for the whole set of Sensitive Privilege Use requirements.

```
abstract public class PrivilegeUseRequirement
    extends AuditPolicyRequirement

abstract public class
    SensitivePrivilegeUseRequirement extends
    PrivilegeUseRequirement
```

With the abstraction hierarchy in place, we can now present the final, specific version of the requirements specifications:

```
public class V_63483 extends
    SensitivePrivilegeUseRequirement
```

Moreover, STIG rule V-63483 has the following parameters:

```
Finding ID: V-63483
Version: WN10-AU-000110
Severity: Medium
STIG: Windows 10 Security Technical
    Implementation Guide
Date: 2019-01-04
Check Text: C-64235r1_chk
Fix Text: F-69413r1_fix
```

So that, these parameters are presented as attributes of the Java class attributes.

```
public class V_63483 extends
    SensitivePrivilegeUseRequirement {
   @Override
   protected String getFailure() {
      return "enable"; }

   @Override
   protected String getInclusionSetting() {
      return "Failure"; }

   @Override
   protected String getSuccess() {
      return null; }

   @Override
   public String checkTextCode() {
      return "C-64235r1_chk"; }

   @Override
   public String date() {
      return "2019-01-04"; }

   @Override
   public String findingID() {
      return "V-63483"; }

   @Override
   public String fixTextCode() {
      return "F-69413r1_fix"; }

   @Override
   public String iAControls() {
      return ""; }

   @Override
   public String ruleID() {
      return "SV-77973r1_rule"; }

   @Override
   public String sTIG() {
      return "Windows 10 Security Technical
          Implementation Guide";
   }

   @Override
   public String severity() {
      return "Medium"; }

   @Override
   public String version() {
      return "WN10-AU-000110"; } }
```

This class is a simple instantiation of the superclass with specific parameters that determine particular behavior for enforcement and verification of V_63483 STIG requirements. This example of how pattern can verify specific requirement. By configuring superclass (in our case, class *Requirement*), we can simplify verification of children classes by updating attributes. More STIG rules implemented as Java classes are stored in GitHub [4].

IV. Discussion

We can highlight the following benefits behind applying object-oriented principles for requirements description:

- Elimination of redundancy with the generalization and inheritance.

[3]https://github.com/anaumchev/VDO-Patterns/blob/master/src/rqcode/stigs/win10/V_63463.java

[4]https://github.com/anaumchev/VDO-Patterns/tree/master/src/rqcode/patterns/win10

- Enhancement of maintainability. When we need to update requirements, we have to configure the corresponding superclass. Given the example above, one may support only one implementation of the SensitivePrivilegeUseRequirement class instead of two, which makes it simpler for maintenance, reuse.

- Extensibility improvement: we can for instance create new requirements for Privilege Use events by extending the PrivilegeUseRequirement class and all the natural language description templates and core checking and enforcing logic would be already implemented.

- Combination of several representations of a requirement and OO logic in one source: it is possible to add representations in other formal notations such as Linear Temporal Logic (LTL) and Timed Computation Tree Logic (TCTL) for further formal verification of requirements. LTL plays the role of baseline for requirements specification representation via RQCODE. A LTL formula which represents the possible path of an event can be presented in RQCODE via Java code or in the form of test cases.

- OO analysis enabled: one can run OO analysis methods to conduct evaluation of the requirements' specification. For example, with a class coupling and depth of inheritance, one can argue about the maintainability and complexity of the requirements' specification. Improved traceability: The ultimate race in the Requirements Engineering domain is the ability to verify and validate various properties while keeping trace to the source requirements. The proposed approach helps to drastically simplify the traceability, since the verification/validation means are incorporated within the requirement specification.

- Integration with Dev tools: one of the major barriers for adoption is availability and integration with the development environments. Each development project selects a particular set of IDEs, configuration management, continuous integration tools and the requirements engineering approaches have to cope with the constraints of those environments. Instead, RQCODE seamlessly integrates with the most common environments, since requirements classes are of the same nature as other source code development artifacts. To illustrate, the configuration management tools, versioning, continuous integration, automated verification and deployment tools for Java will be directly applicable to requirements as code.

- Integration with Dev processes: the modern development process heavily relies on management tools such as GitHub, GitLab and others that naturally focus on source code with issue tracking, tasks planning and tracking, automated notifications, etc. The current concept of requirements management will seamlessly integrate with development processes.

- Integration with Quality Assurance process: below we discuss the relationship between RQCODE and Test Driven Development (TDD). First we specify that we consider TDD as an advanced technique consisting in using automated unit tests to drive software design and force dependencies decoupling [10]. RQCODE and TDD have common characteristics: they both can be started before code implementation based on requirements' specification. Our work resembles TDD approach as we also suggest test cases as requirements. However, in contrast to TDD, we do not impose any process and we concentrate on reuse of requirements, what we believe is more applicable for security properties. The principles of RQCODE guarantee the reduction of code duplication and the simplification of code maintenance, on the contrary, the main drawback of TDD is that the test cases duplicate the amount of code to be written and maintained.

The goal of the thesis work briefly outlined in this paper will be to explore the applicability of the object-oriented requirements for the security requirements domain. We have identified the following challenges for the work:

- Software Engineering standards such as ISO 25010 [3] define several categories of security quality attributes. Will SOOR be effective for specifying each of those categories?

- While the SOOR concepts have been explored for a while, it is still under exploration even for the functional requirements.

- Security requirements are particular since they may be functional and non-functional, positive and negative. Will SOOR be equally effective for all those types of requirements?

- Many security standards and guidelines are expressed on a very high level (maybe an example from IEC 62443 [11] without clear instructions for possible verification means.

- Security testing is a vast domain, as outlined in [4]. We need to assess the ability of SOOR and RQCODE in particular to accommodate this variety of verification means.

- The applicability should be analyzed through industrial case studies, since the practice is one of the best measures of validity in Software Engineering. Will SOOR be effective in case studies from the VeriDevOps project [12]?

V. Related work

The proposed work touches upon several related topics that we plan to analyze and extend. Firstly, our work has to take advantage of the best practices outlined by [13] and [14]. Secondly, the work has considered existing methods for requirements formalization as described in [15]. The proposed research deals with requirements modeling with OOP concepts and representation of requirements in several notations at once. This is addressed by the important work in [16], [17], [18]. The requirements traceability improvements proposed by RQCODE have to be analyzed regarding [19], [20]. Finally, the work at [21] set the basis for our work.

VI. Conclusion

In this positional paper, we outlined a topic of Requirements as Code (RQCODE) as an implementation of Seamless Object-Oriented Requirements (SOORs) for Software Security domain. For the future work, we propose to explore the applicability of SOOR to Software Security domain by systematically analysing the security properties categories and implementing standard requirements in RQCODE. The results will be assessed in application to case studies in the VeriDevOps project.

Acknowledgment

This work has received funding from Horizon 2020 program under the grant agreement No. 957212 – VeriDevOps project.

REFERENCES

[1] B. Boehm, "A view of 20th and 21st century software engineering," in *Proceedings of the 28th international conference on Software engineering*, ser. ICSE '06. New York, NY, USA: Association for Computing Machinery, May 2006, pp. 12–29. [Online]. Available: https://doi.org/10.1145/1134285.1134288

[2] B. Mburano and W. Si, "Evaluation of web vulnerability scanners based on OWASP benchmark," in *2018 26th International Conference on Systems Engineering (ICSEng)*, 2018, pp. 1–6.

[3] ISO/IEC 25010:2011, SQuaRE — system and software quality models. [Online]. Available: https://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/03/57/35733.html

[4] M. Felderer, M. Büchler, M. Johns, A. D. Brucker, R. Breu, and A. Pretschner, "Chapter one - security testing: A survey," in *Advances in Computers*, A. Memon, Ed. Elsevier, 2016, vol. 101, pp. 1–51. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0065245815000649

[5] R. N. Rajapakse, M. Zahedi, M. A. Babar, and H. Shen, "Challenges and solutions when adopting DevSecOps: A systematic review," vol. 141, p. 106700, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584921001543

[6] N. Messe, V. Chiprianov, N. Belloir, J. El-Hachem, R. Fleurquin, and S. Sadou, "Asset-oriented threat modeling," in *TrustCom 2020-19th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*. IEEE, 2020, pp. 1–11.

[7] M. Vachharajani, N. Vachharajani, and D. I. August, "A comparison of reuse in object-oriented programming and structural modeling systems," p. 5.

[8] K. Ismaeel, A. Naumchev, A. Sadovykh, D. Truscan, E. P. Enoiu, and C. Seceleanu, "Security requirements as code: Example from VeriDevOps project," in *2021 IEEE 29th International Requirements Engineering Conference Workshops (REW)*, 2021, pp. 357–363.

[9] A. Naumchev, "Exigences orientées objets dans un cycle de vie continu," phdthesis, 2019. [Online]. Available: http://thesesups.ups-tlse.fr/4468/

[10] D. Duka and L. Hribar, "Test Driven Development Method in Software Development Process," p. 5.

[11] International Electrotechnical Commission, International Electrotechnical Commission, and Technical Committee 65, *Security for industrial automation and control systems*, 2017, OCLC: 1024071994.

[12] A. Sadovykh, G. Widforss, D. Truscan, E. P. Enoiu, W. Mallouli, R. Iglesias, A. Bagnto, and O. Hendel, "VeriDevOps: Automated protection and prevention to meet security requirements in DevOps," in *2021 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2021, pp. 1330–1333, ISSN: 1558-1101.

[13] S. Hansen, N. Berente, and K. Lyytinen, "Requirements in the 21st century: Current practice and emerging trends," in *Design requirements engineering: A ten-year perspective*. Springer, 2009, pp. 44–87.

[14] S. A. Fricker, R. Grau, and A. Zwingli, "Requirements engineering: best practice," in *Requirements Engineering for Digital Health*. Springer, 2015, pp. 25–46.

[15] J.-M. Bruel, S. Ebersold, F. Galinier, M. Mazzara, A. Naumchev, and B. Meyer, "The role of formalism in system requirements," *ACM Comput. Surv.*, vol. 54, no. 5, may 2021. [Online]. Available: https://doi.org/10.1145/3448975

[16] I. Jacobson, I. Spence, and K. Bittner, *USE-CASE 2.0 The Guide to Succeeding with Use Cases*. Alexandria, Virginia: Ivar Jacobson International SA., Dec. 2011. [Online]. Available: https://www.ivarjacobson.com/sites/default/files/field_iji_file/article/use-case_2_0_jan11.pdf

[17] B. Meyer, "Multirequirements," in *Modelling and Quality in Requirements Engineering: Essays dedicated to Martin Glinz on the occasion of his 60th birthday*. Verl.-Haus Monsenstein u. Vannerdat, 2013.

[18] F. Galinier, "Seamless development of complex systems: a multirequirements approach," Ph.D. dissertation, Paul Sabatier University, Toulouse, France, 2021.

[19] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo, "Recovering traceability links between code and documentation," *IEEE Trans. on Software Engineering*, vol. 28, no. 10, pp. 970–983, 2002.

[20] J. Cleland-Huang, "Toward improved traceability of non-functional requirements," in *Proc. of the 3rd international workshop on Traceability in emerging forms of software engineering*, 2005, pp. 14–19.

[21] A. Naumchev, "Seamless object-oriented requirements," in *2019 International Multi-Conference on Engineering, Computer and Information Sciences (SIBIRCON)*, 2019, pp. 0743–0748.