VeriDevOps Software Methodology: Security Verification and Validation for DevOps Practices

Eduard Paul Enoiu
Mälardalen University
Västerås, Sweden
eduard.paul.enoiu@mdu.se

Dragos Truscan
Åbo Akademi University
Turku, Finland
dragos.truscan@abo.fi

Andrey Sadovykh
SOFTEAM
Paris, France
andrey.sadovykh@softeam.fr

Wissam Mallouli
Montimage
Paris, France
wissam.mallouli@montimage.com

ABSTRACT

VeriDevOps offers a methodology and a set of integrated mechanisms that significantly improve automation in DevOps to protect systems at operations time and prevent security issues at development time by (1) specifying security requirements, (2) generating trace monitors, (3) locating root causes of vulnerabilities, and (4) identifying security flaws in code and designs. This paper presents a methodology that enhances productivity and enables the continuous integration/delivery of trustworthy systems. We outline the methodology, its application to relevant scenarios, and offer recommendations for engineers and managers adopting the VeriDevOps approach. Practitioners applying the VeriDevOps methodology should include security modeling in the DevOps process, integrate security verification throughout all stages, utilize automated test generation tools for security requirements, and implement a comprehensive security monitoring system, with regular review and update procedures to maintain relevance and effectiveness.

KEYWORDS

security, verification, testing, monitoring, DevOps

1 INTRODUCTION

In today's fast-paced software development world, the need for security is more crucial than ever before. However, security is often seen as an afterthought, and it is only addressed after an application has been deployed, making it difficult and costly to fix any issues. This is where DevOps [9] comes into play since it helps to combine

the speed and agility of software development with the reliability and security of operations.

SecDevOps [11] is a software development methodology that merges DevOps practices with security principles, aiming to establish a seamless and cohesive approach to software development and security. The primary objective of SecDevOps is to incorporate security measures at every phase of the software development lifecycle (SDLC), encompassing planning, design, deployment, and maintenance stages. In traditional software development, security is typically treated as a separate and isolated step within the SDLC [10]. However, this approach often results in discovering security vulnerabilities late in the process, leading to delays, increased expenses, and compromised security. To overcome these challenges, SecDevOps advocates for the integration of security throughout the entire SDLC. This approach emphasizes collaboration, automation, and continuous monitoring to ensure that security is an intrinsic element of the software development process. By adopting SecDevOps, organizations can proactively address security concerns, minimize risks, and enhance the overall security posture of their software applications. SecDevOps is increasingly being embraced to enhance the security of software applications. Through the seamless integration of security throughout the SDLC, SecDevOps empowers organizations to deliver secure software in a faster and more efficient manner.

To integrate security into the DevOps process effectively, it is recommended to employ security verification, automated security verification, and security monitoring techniques. Security verification [3] is a method of demonstrating that a system or application is secure through assurance and reasoning. Automated security testing [5] involves using software tools to automatically generate and execute tests in order to detect any security vulnerabilities in an application. Finally, security monitoring [13] involves continuously tracking an application's performance and behaviour to detect any attacks or potential security breaches. Together, these methods make it possible for the *VeriDevOps methodology* to provide a comprehensive approach to security in DevOps, enabling teams to identify and address security issues throughout the development process rather than after deployment. In this methodology, security is integrated into every stage of the DevOps pipeline, from planning and development to testing, deployment and monitoring.

The VeriDevOps methodology combines multiple techniques to embed security throughout the DevOps lifecycle. It begins by using

Natural Language Processing (NLP) to improve security requirements, eliminating poorly written specifications and enhancing their understandability. Modern NLP methods are further leveraged to extract, classify, and semantically search related security recommendations from natural language texts. A requirements-as-code paradigm is implemented to encapsulate requirements and their verification methods within object-oriented classes, enabling executable requirements, traceability, and reuse. To detect security incidents, monitoring tools analyze network traffic and system logs. Real-time detection tools enhance this process, inspecting network traffic at the packet level with pre-trained neural network models for the swift identification of threats. At the development level, metamorphic testing is employed when systems lack explicit test oracles, assessing the impact of input transformations on outputs. Lastly, specific methodologies are used to undergo validation and generate regression test cases. Collectively, these techniques create a comprehensive, secure, and efficient DevOps process.

The main innovation in VeriDevOps is the direct use of security requirements written in natural language for security test automation and monitoring. The ability to convert these natural language security requirements into automated testing and monitoring rules enables more seamless and continuous integration of security within the DevOps lifecycle. Finally, using natural language for security requirements fosters continuous feedback and learning opportunities. By observing the outcomes of the automated tests and monitoring, teams can improve and refine their security requirements, leading to a more mature, robust, and secure system over time.

2 RELATED WORK

As an example of a technique that can be used in DevOps security-related practices, Kulik et al. [8] explore the use of formal methods in addressing cyber-security concerns in critical infrastructure systems. It reviews three main classes - theorem proving, model checking, and lightweight formal methods - and categorizes them based on the type of computing systems they are applied to. The article presents and compares solutions within each class and category, discussing historical highlights and developments. It offers a comprehensive overview of formal methods and techniques for security-critical systems, simplifying the tool selection process for system designers.

Another technique that can be used in a DevOps methodology is model-based security testing (MBST) [16], which is a relatively new field dedicated to systematically and efficiently specifying and documenting security test objectives, test cases, and test suites, while also automating or semi-automating their generation. This paper presents a survey on MBST techniques, including security functional testing, model-based fuzzing, risk- and threat-oriented testing, and the use of security test patterns.

Granata et al. [6] address the complexity of software systems due to expanding IT infrastructures and the accompanying challenges in managing software vulnerabilities and cybersecurity issues. Even though security-oriented methodologies like SecDevOps have emerged to identify security problems early in the software development life cycle, their integration into standard or custom software development life cycles, as well as, design evaluation and risk management methodologies, remains difficult. To alleviate this, Granata et al. propose MetaSEnD, a new meta-model that outlines the main activities in a Secure Software Development Life Cycle (SS-DLC), and demonstrate its application in a continuous integration pipeline of a sample microservices application.

These approaches incorporate security principles into software development but leave certain aspects unaddressed. In traditional DevOps practices, there exists a critical disconnect where automation and security are not integral to the DevOps lifecycle - they are often treated as separate processes. This forms a gap that can lead to vulnerabilities in the software system. Furthermore, reliance on manual security testing and monitoring methods can lead to inefficiencies and potentially undetected vulnerabilities, further expanding the gaps. VeriDevOps [15] fills these gaps by formalizing security requirements, generating trace monitors, locating root causes of vulnerabilities, and identifying security flaws in code and designs. VeriDevOps focuses on improving automation in DevOps for system protection, offering a more comprehensive approach to building secure, trustworthy systems efficiently and effectively.

3 VERIDEVOPS METHODOLOGY

VeriDevOps is a methodology that combines the principles of DevOps and early verification, test automation and monitoring to ensure that software systems are secure and reliable. An overview is shown in Figure 1. It aims to provide a systematic approach to software development that ensures that security requirements are met throughout various stages of software development. The methodology involves the use of Natural Language Processing (NLP), specification of security requirements, automatic quality assurance to verify the correctness of software systems and the use of DevOps practices to ensure that software systems are developed and deployed securely and reliably.

VeriDevOps automates essential components of software development with security considerations. This includes the specification and analysis of security-related requirements, system testing and monitoring, and the incorporation of these tools and techniques into prevailing VeriDevOps practices in the industry.

The process starts with the analysis and formalization of a text-based description of security requirements sourced from various inputs. NLP and pattern/boilerplate usage are key technologies in preventing the introduction of inconsistencies or ambiguities into the specification. Additionally, methods for auto-translating patterns into temporal logic are utilized.

Another vital aspect is the automated configuration of trace monitors, founded on the specification of security requirements using semi-structured or structured formalisms. These traces are configured over time and monitored continually using these modelled specifications. Runtime monitoring is employed to identify anomalies and vulnerabilities.

Moreover, we auto-generate attack tests based on the specification of security requirements. This includes identifying invalid states and contradictory security requirements in natural language requirement specifications and using this data to create negative tests. These tests aim to push the system into invalid states, i.e. potentially insecure behavior, thereby uncovering potential vulnerabilities that positive test cases may not detect. One enhancement
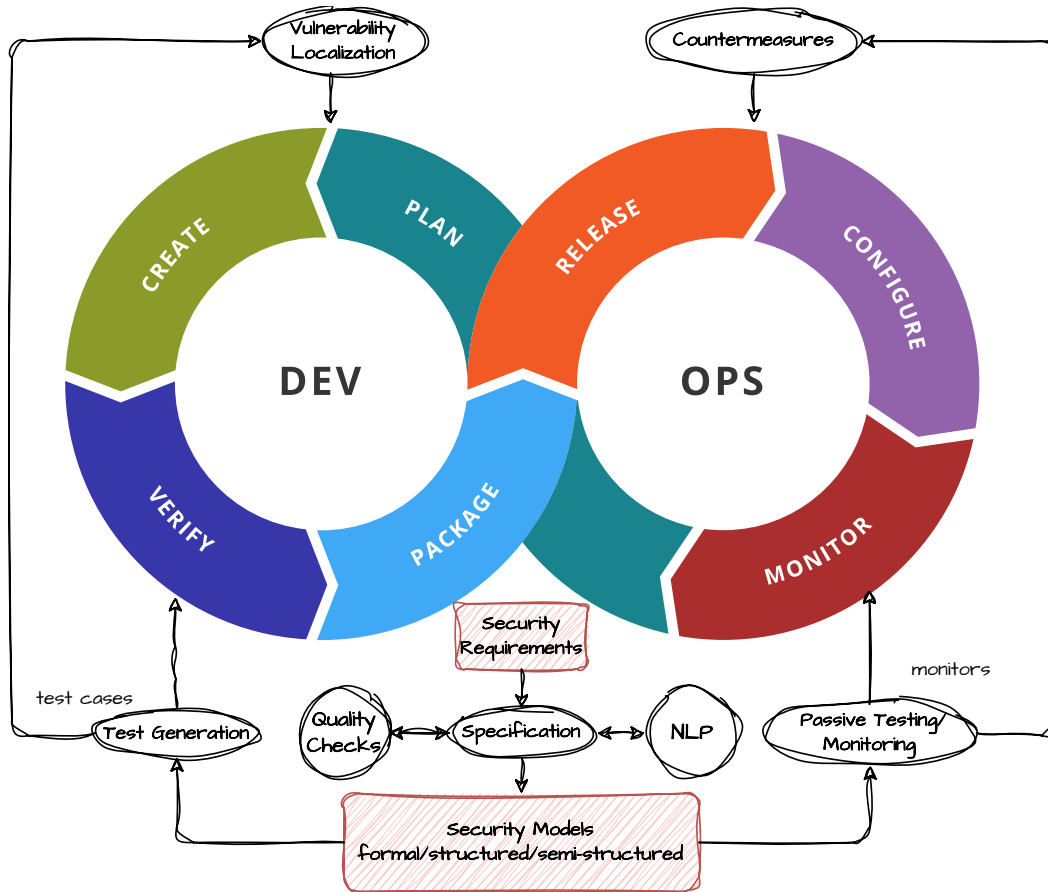
Figure 1: An Overview of the VeriDevOps Methodology

could involve establishing guidelines and a format for testers to propose test scenarios that evaluate both security and energy properties, areas often overlooked in testing.

Lastly, VeriDevOps automates design and code checks according to the specification of security requirements using semi-structured and structured formalisms. These verification activities can be conducted either by simulating the resulting model or by formally verifying a system description.

VeriDevOps Methodology contains a set of comprehensive guidelines for the adopters of security verification methods and tools and can also serve as teaching material for IT-oriented universities. The methodology guides users in applying security requirements for automated generation, protection, and prevention activities driven by security specifications.

3.1 VeriDevOps Architecture

The VeriDevOps Methodology groups several interconnected tool sets for Security Requirements Generation, Reactive Protection at Runtime as well as for Prevention at Design and Development (as shown in Figure 2). Those tool sets are highly interconnected to achieve the goal of linking security requirements with design analysis, verification at the code level and the runtime analysis of

systems. The VeriDevOps tool sets mix concrete tool components provided and developed by VeriDevOps partners. These concrete tools differ in licensing policies and maturity. While there are many mature commercial and open-source tools, others are more experimental. These tools should implement the interfaces and features of the VeriDevOps Methodology and should be interchangeable to a certain extent. The case studies will employ a mix of those tools that better correspond to the requirements and will fit better with the methodology or industry practice.

3.1.1 Requirements Specification. The VeriDevOps process involves the examination and formalization of security requirements, which are gathered from a range of textual descriptions. To avoid introducing inconsistencies and ambiguities into these specifications, we utilize NLP and established patterns or boilerplates. Additionally, we employ techniques that automatically translate these patterns into temporal logic, further enhancing the clarity and consistency of our security requirements. Several techniques can be implemented in the VeriDevOps methodology, such as PROPAS and RQCode. In addition, both manual and semi-automatic translation methods can also be utilized to optimize the requirements formalization. In addition, verification and analysis tasks can be performed either by simulating the final model or by verifying the system's
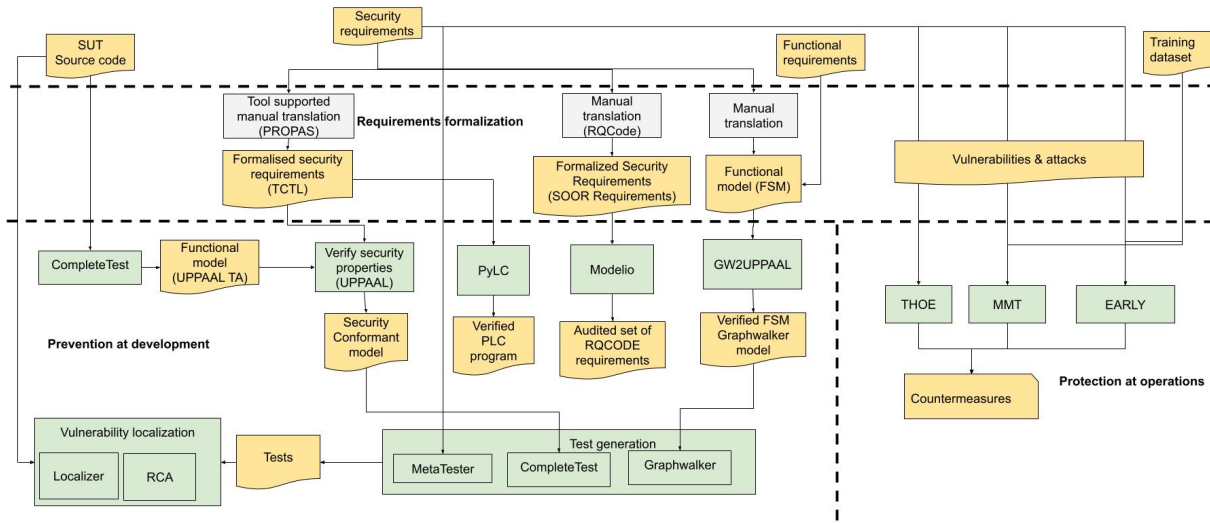
Figure 2: An Overview of the VeriDevOps Methodology Architecture and Examples of Tool Components

description. Based on several natural languages and model smells, we established a set of indicators (i.e., using NALABS) for security requirement flaws and defined metrics to automatically detect these smells in security artefacts.

3.1.2 Prevention at Development. Several techniques are used during this phase for test modelling (e.g., UPPAAL, PyLC, Modelio, GW2UPPAAL), automated test generation (e.g., MetaTester, CompleteTest, Graphwalker) and vulnerability localization (e.g., Localizer, RCA). This information assists in generating positive and negative tests designed to push the system into certain states, thereby exposing potential vulnerabilities. To enhance this process, it would be beneficial to establish guidelines and a format that enables testers to design test scenarios that assess not only security aspects but also energy properties, as these are often overlooked areas in testing.

3.1.3 Protection at Operations. We deploy an automated setup of monitoring tools (e.g., MMT, THOE, EARLY), which is based on the specification of security requirements in natural language, semi-structured or structured formalisms. Over time, these traces are automatically configured and continually observed using formal or semi-formal specifications. Runtime monitoring, a technique that observes system behavior during operation, is implemented to detect errors, monitor performance, ensure compliance, and maintain system health, thereby providing a basis for potential preemptive countermeasures.

4 AN INSTANTIATION OF THE VERIDEVOPS METHODOLOGY

In this section, we illustrate the VeriDevOps methodology by selecting several tools for requirements specification, verification, and monitoring as depicted in Figure 3. We first describe the functionality of these tools; then we exemplify their applicability with concrete examples.

4.1 A Selection of Tools for the VeriDevOps Methodology

4.1.1 NALABS: Detecting Bad Smells in Security Requirements. Natural language is often used to express requirements and test specifications in large-scale embedded system development. However, in such contexts, requirement review is often done manually, which can be a time-consuming and error-prone process. Poor-quality specifications can have costly consequences on the requirement engineering process, particularly if feedback loops are lengthy, resulting in artifacts that are difficult to maintain, understand, and transfer to other system variants. To address this issue, NALABS [14] (NAtural LAnguage Bad Smells) tool utilizes the concept of "smells" to identify bad specifications related to safety and security requirements. By utilizing NALABS, DevOps engineers can improve the quality and maintainability of their natural language specifications, resulting in more efficient and effective requirement engineering processes. In the development process, the quality of safety and security requirements expressed in text form is critical. Poor-quality requirements can result in hard-to-detect errors or those identified too late. NALABS can assist engineers in writing high-quality specifications, ultimately improving the overall quality of the security requirements.

4.1.2 ARQAN: Automated Security Requirements Analysis with NLP. ARQAN[1] is the security analysis tool based on the modern NLP methods such as Large Natural Language Models, Transfer Learning and BERT as exemplified in [7]. The main tasks for ARQAN are to extract security requirements from texts in Natural Language, classify the requirements as well as semantic search for related recommendations in Security Technology Implementation Guidelines (STIG) database.

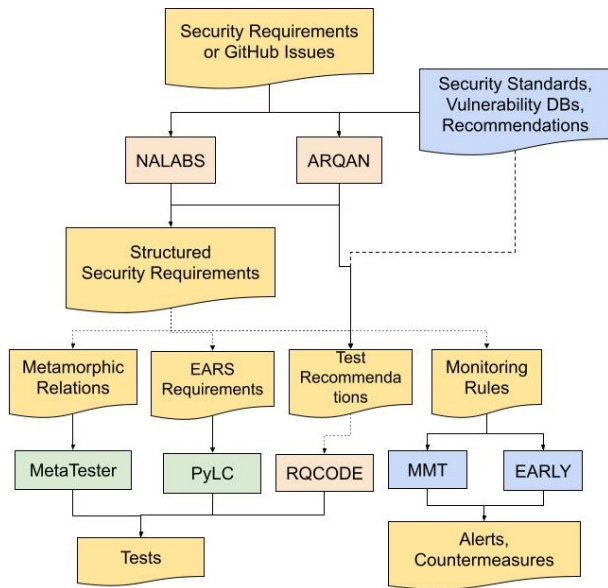[1]The ARQAN prototype is available at https://arqan.softeam-rd.eu:8501/

Figure 3: Selected tools to exemplify the VeriDevOps methodology

4.1.3 RQCODE: Requirements as Code. RQCODE[2] is a novel approach that applies the Seamless Object-Oriented Requirements (SOOR) paradigm to be implemented in Java language. The RQCODE approach represents requirements as classes that contain various representations, including the textual one: requirements description in natural language as well as methods for verifying these requirements, such as an acceptance test. This allows for direct traceability between a requirement and its implementation that can be checked at any time through the execution of the included test. Moreover, object-oriented implementation supports easy reuse of requirements and tests by the standard means, such as inheritance, provided by the language, e.g., Java. One requirement can be an extension or a specialisation of another one. Each requirement can be considered as a template for requirements of a similar kind, e.g. by initialising a requirement class with different parameters.

RQCODE includes specific packages for the baseline concepts and temporal patterns since they represent the foundation for writing requirements using well-known templates. For experimentation purposes, we prototyped examples for Windows 10 and Ubuntu 18 related Security Technology Implementation Guides (STIG) [12].

4.1.4 MMT: . Montimage Monitoring Tool (MMT) is a monitoring solution that allows capturing and analyzing network traffic. It can be used to understand how the network is used (protocols, applications, and users) and detect potential security and performance incidents. The MMT tool has a plugin architecture that allows its easy extension in order to target new protocols and, more generally, any input data to be analyzed. In the context of VeriDevOps, MMT will be extended to analyse system and application logs provided by industrial case studies, to check a set of predefined security properties to detect security incidents. It will also integrate an anomaly

detection mechanism that relies on ML/AI algorithms to identify potential drifts and deviations from a learned baseline. A root cause analysis to determine the origins of such security incidents and anomalies will allow us to better select the countermeasure that can be applied among a set of potential resilience catalogues.

4.1.5 EARLY. [1, 2] is a tool for real-time detection of ongoing security attacks against a target by monitoring the incoming and outgoing traffic at the packet level. The tool identifies network flows (related sequences of network packets) in the network traffic and classifies them as normal or malicious. The innovation in EARLY comes from the fact that the tool can detect network attacks in real-time, by inspecting only a portion of the packets, before the attack completes. This allows to deploy countermeasures effectively and efficiently. The engine of the monitoring tool uses neural network models pre-trained for recognizing security attacks in different application domains. The EARLY can be used either to detect known attacks or anomalies in the network traffic. The tool uses a library of pre-trained neural network models corresponding to different application domains.

4.1.6 Meta-Localizer. is an integrated tool-supported approach consisting of two modules. The first one, MetaTester [17], is used for testing systems with a non-explicit test oracle. Metamorphic testing (MT) was introduced as a solution to test systems without explicit specifications of the test oracle. In MT, the behavioral or functional properties of the system are defined by posing a hypothesis about using generic relations known as metamorphic relations (MRs) between different sets of inputs and their expected outputs. An MR is composed of two parts: an input relation and an output relation. An input relation represents the relation between the inputs of the source and follow-up test cases, whereas an output relation represents the relation between the expected outputs of the source and follow-up test cases. A source test case is the first set of tests performed using seed inputs. The seed inputs are transformed into morphed inputs. The follow-up test cases are performed using these morphed inputs. In addition, an implication between the outputs of the source and follow-up test cases is needed to specify the impact of input transformations on their corresponding outputs. To that extent, the tool uses metamorphic relations (MRs) extracted from security requirements and uses them for assigning test verdicts. The second module, the Localizer, uses the passed and failed tests produced by the MetaTester and identifies the piece of code of the system under test that is more probable to be the source of the vulnerability.

4.1.7 PyLC [4]: is a method developed for converting PLC (Programmable Logic Controller) programs into Python code. This method incorporates three separate unit-testing validation mechanisms to verify the translation process. Our evaluation of this method, applied to various industrial PLC programs, demonstrates its effectiveness and efficiency. The ultimate goal is to use PyLC to produce search-based test cases for PLC programs to be utilized in regression testing during the development of industrial control systems.

4.1.8 Additional Tools. The VeriDevOps methodology utilizes a diverse set of tools for assuring the security and quality of software.

---

[2]RQCODE is freely available at https://github.com/VeriDevOps/RQCODE

Figure 4: Automating Security Requirements Analysis with ARQAN

For a complete list of tools, we refer the reader to the report on the architecture and implementation evaluation available [3].

4.2 Illustrative Examples

In modern software development practices, requirements are specified in natural language. In DevOps practice, requirements are registered as feature request issues in an issue tracking system. The developer's team analyses the feature requests and schedules them in a backlog - a detailed list of tasks, often with an assignment to a concrete responsible developer.

However, manual analysis of requirements may lead to misinterpretation or an assignment to the wrong person or team. The security requirements that come from standards such as IEC 62443, NIST SP 800-53 or even OWASP Application Security Verification Standard may be very generic. This leaves developers without clear guidelines for implementing and verifying security measures. This may lead to inadequate security protection implemented in the IT system.

Consider the following generic requirement: *The system should be protected from brute force attack on ssh*. The developer has to correctly interpret this requirement, implement the protection measures, and verify that the implementation satisfies this requirement.

This may be quite complicated since the requirement is very general. VeriDevOps proposes to integrate the automated recommending tools that would analyse the requirement for quality (i.e., NALABS) and possibly provide guidelines (i.e., ARQAN) from the existing repository, such as Security Technology Implementation Guidelines (STIG).

The following concrete recommendation was detected with the semantic search by ARQAN for the Ubuntu operating system:

- **V-219313.** *The Ubuntu operating system must use SSH to protect the confidentiality and integrity of transmitted information unless otherwise protected by alternative physical safeguards, such as, at a minimum, a Protected Distribution System (PDS).*
- **V-219308.** *The Ubuntu operating system must enforce SSHv2 for network access to all accounts.*
- **V-219310.** *The Ubuntu operating system must immediately terminate all network connections associated with SSH traffic after a period of inactivity.*
- **V-219306.** *The Ubuntu operating system must monitor remote access methods.*
- **V-219265.** *The Ubuntu operating system must generate audit records for successful/unsuccessful uses of the chsh command.*

VeriDevOps suggests integrating these methods in the Continuous Integration and Deployment (CI/CD) pipelines so that its

---

[3]https://www.veridevops.eu/veridevops/download/deliverables

Figure 5: Examples of security requirements handled by NALABS. Detected findings are highlighted and categorized according to their metrics.

requirements are correctly classified as related to security as soon as possible. This would help to assign an appropriate response and provide relevant guidelines. For example, in Figure 4, a security requirement was added to the GitHub issue tracker. The system has correctly classified this requirement as related to security and added the corresponding label. Next, a set of relevant Security Implementation Guides (STIGs) was automatically provided with the semantic search method. The recommendation provides practical instructions on testing the security requirement and fixing possible issues. The system also browses the repository of the implemented RQCODE requirements tests for the identified STIGS. It is possible to import the existing RQCODEs to the own repository or to create a feature request for the necessary but missing STIG.

In addition, the same requirement for SSH brute force protection can be enforced by deploying and configuring MMT and EARLY for detecting anomalies when the SSH service is accessed, as will be discussed later.

NALABS focuses on utilizing bad smells to identify specification quality defects in industrial settings. NALABS interface (as shown in Figure 6) examines the quality of natural language specifications, the creation of smell metrics, and the automated measurement of these smells using specific dictionaries. Key aspects explored include vagueness, referenceability, optionality, subjectivity, weakness, readability, and complexity metrics. These metrics provide insights into issues such as understanding requirements, nesting and referencing in documents, use of optional words, presence of subjective language, the introduction of uncertainty, readability scores, and the size and complexity of requirements.
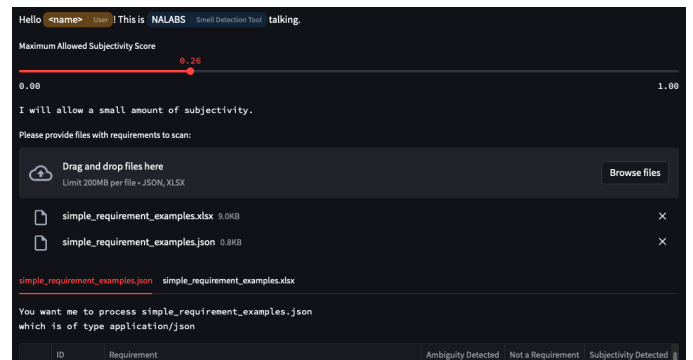

Figure 6: Automating Security Requirements Bad Smells Detection with NALABS quality checker.

We show in Figure 5 several security requirements. For example, two of the requirements include optionality smells (e.g., the use of the "can" and "may" words).

The MetaTester tool is used to generate metamorphic tests and verify their conformance with the metamorphic relations. For example, for the security requirement extracted previously: *"System should be protected from brute force attack on SSH"*, we can define a metamorphic relation, such as "Authenticating with wrong credentials should result in the same response." In this case, tests are generated for different combinations of username and password. Instead of checking the output of each test, we check that all the outputs are equal. In case the latter condition does not hold, it

Figure 7: EARLY tool screen caption when detecting a brute force attack.

Figure 8: The generated test cases based on the EARS syntax as shown in CODESYS IDE.

means that a successful authentication has occurred. The MetaTester has also been applied to Load Position System (LPS) module in an industrial case study to test whether the markers of the load are properly identified in the presence of environmental (e.g., water) reflections. In this case, the metamorphic tests were generated and verified with the following metamorphic relation: "*The same correct markers of the load should be correctly identified both in the absence and in the presence of reflections from the environment*". From the execution of the tests, we were able to identify two classes of incorrect classifications false positives – the system identified the reflections as the real ones and false negatives – the system did not identify any markers in the presence of reflections although the former were present in the input. More details on the approach and its results can be found in [17].

The same security requirement is used for monitoring the system against SSH brute force attacks. Based on this requirement, the monitor component of Early is configured to use the model trained for SSH-related attacks, including the brute force attack. During the monitoring process, whenever such an attack is detected, an alarm is automatically triggered to deploy countermeasures, e.g., blocking of the remote host (as shown in Figure 7).

We show the combination of two techniques, EARS and PyLC, for addressing the challenges of requirement formalization and automated test generation. The EARS approach focuses on formalizing natural language requirements using a semi-structured syntax. At the same time, PyLC is a framework that integrates Pynguin, a search-based testing tool for Python, to generate test cases for PLC programs automatically. Our results demonstrate the applicability of EARS in formalizing security requirements and developing test cases for PLC programs to find vulnerabilities. Furthermore, the generated test cases are executed using CODESYS Test Manager, ensuring consistency between the Python and original PLC environments. We provide a snippet of the generated test cases based on the EARS syntax in Figure 8. Finally, further improvements, such as exploring alternative search-based algorithms and allocating larger test generation budgets, are suggested to enhance test coverage and mutation analysis, particularly for more extensive PLC programs.

5 RECOMMENDATIONS FOR PRACTITIONERS

Here are some recommendations that practitioners can use when applying the VeriDevOps methodology for security verification, automated testing, and monitoring in DevOps:

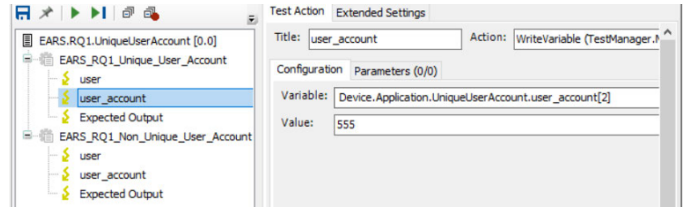**Enhancing DevOps Security Through Effective Management of Security Requirements:**

- Incorporate specifications for security requirements into the DevOps process by using both semi-structured and structured formal methods. This will help to guarantee that security requirements are effectively handled. A variety of VeriDevOps tools [4] are available that can facilitate this process of expressing requirements.
- Ensure that the security requirements are sufficiently clear and detailed to enable verification. For example, several VeriDevops tools (e.g., [12, 14]) can be used to streamline the requirement writing process, making it more consistent and efficient, but also providing immediate feedback, aiding in refining requirements.

Integrating Security Requirements Analysis and Verification into the Entire DevOps Process:

- Use bad smells detectors (e.g., [14]), NLP and security requirements patterns (e.g., [12]) to specify and analyze security requirements in short feedback loops. Ensure that security requirements are specified clearly and unambiguously regardless of the formalism used e.g., natural language, semi-structured patterns, models.
- Use automated tools (e.g., ARQAN, RQCODE) to validate the security models against security requirements.

Using Automated Tools to Generate and Select Security Tests:

- Integrate security testing into the DevOps pipeline. Several tools for active prevention, such as PyLC[4], which have applications ranging from security test generation to combinatorial security testing.
- Adopt different testing techniques, such as metamorphic testing, for systems without an explicit oracle. In our VeriDevOps approach, we proposed the use of MetaTester. [17]

Implementing security monitoring systems to detect and respond to security threats:

- Monitor all critical components of the DevOps pipeline, including code repositories, build servers, and deployment environments.
- Use automated tools (e.g., EARLY [2]) to detect anomalies and security attacks. Regularly review and update the security monitoring system to ensure it remains effective.

---

[4]More details on these VeriDevOps tools can be found at https://cordis.europa.eu/project/id/957212/results

6 CONCLUSIONS

In conclusion, security verification, automated testing, and monitoring are essential methods in the DevOps process for ensuring the security and reliability of software applications. By incorporating these methods into VeriDevOps methodology, teams can detect and address security vulnerabilities at every stage, from planning and development to testing and deployment.

Security verification enables teams to demonstrate the security of an application through automated verification. Automated testing allows for the quick and efficient detection of security vulnerabilities, saving time and resources. And monitoring enables teams to continuously track an application's behaviour and performance, ensuring that any potential security breaches are detected and addressed in real-time.

By utilizing the VeriDevOps methodology, DevOps teams can not only enhance the security of their applications but also improve the overall quality and reliability of their software. It's important to remember that security is not a one-time event but a continuous process that requires ongoing attention and investment. With the integration of security verification, automated testing, and monitoring into the DevOps process, teams can ensure that their applications are secure and reliable, providing peace of mind for both themselves and their users.

ACKNOWLEDGMENTS

REFERENCES

[1] Tanwir Ahmad and Dragos Truscan. 2022. Early GitHub Repository. https://github.com/VeriDevOps/Earlytool
[2] Tanwir Ahmad, Dragos Truscan, Juri Vain, and Ivan Porres. 2021. Early Detection of Network Attacks Using Deep Learning. In *IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSWT)*. IEEE, Online. https://doi.org/10.1109/ICSTW55395.2022.00020
[3] Matteo Avalle, Alfredo Pironti, and Riccardo Sisto. 2014. Formal verification of security protocol implementations: a survey. *Formal Aspects of Computing* 26 (2014), 99–123.
[4] Mikael Ebrahimi Salari, Eduard Paul Enoiu, Wasif Afzal, and Cristina Seceleanu. 2023. PyLC: A Framework for Transforming and Validating PLC Software using Python and Pynguin Test Generator. In *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing*. 1476–1485.
[5] Vahid Garousi and Frank Elberzhager. 2017. Test automation: not just for test execution. *IEEE Software* 34, 2 (2017), 90–96.
[6] Daniele Granata, Massimiliano Rak, and Giovanni Salzillo. 2022. MetaSEnD: A Security Enabled Development Life Cycle Meta-Model. In *Proceedings of the 17th International Conference on Availability, Reliability and Security*. 1–10.
[7] Vladimir Ivanov, Andrey Sadovykh, Alexandr Naumchev, Alessandra Bagnato, and Kirill Yakovlev. 2022. Extracting Software Requirements from Unstructured Documents. In *Recent Trends in Analysis of Images, Social Networks and Texts (Communications in Computer and Information Science)*, Evgeny Burnaev, Dmitry I. Ignatov, Sergei Ivanov, Michael Khachay, Olessia Koltsova, Andrei Kutuzov, Sergei O. Kuznetsov, Natalia Loukachevitch, Amedeo Napoli, Alexander Panchenko, Panos M. Pardalos, Jari Saramäki, Andrey V. Savchenko, Evgenii Tsymbalov, and Elena Tutubalina (Eds.). Springer International Publishing, Cham, 17–29. https://doi.org/10.1007/978-3-031-15168-2_2
[8] Tomas Kulik, Brijesh Dongol, Peter Gorm Larsen, Hugo Daniel Macedo, Steve Schneider, Peter WV Tran-Jørgensen, and James Woodcock. 2022. A survey of practical formal methods for security. *Formal Aspects of Computing* 34, 1 (2022), 1–39.
[9] Leonardo Leite, Carla Rocha, Fabio Kon, Dejan Milojicic, and Paulo Meirelles. 2019. A survey of DevOps concepts and challenges. *ACM Computing Surveys (CSUR)* 52, 6 (2019), 1–35.
[10] M Mahalakshmi and Mukund Sundararajan. 2013. Traditional SDLC vs scrum methodology–a comparative study. *International Journal of Emerging Technology and Advanced Engineering* 3, 6 (2013), 192–196.
[11] Vaishnavi Mohan and Lotfi Ben Othmane. 2016. Secdevops: Is it a marketing buzzword?-mapping research on security in devops. In *2016 11th international conference on availability, reliability and security (ARES)*. IEEE, 542–547.
[12] Ildar Nigmatullin, Andrey Sadovykh, Nan Messe, Sophie Ebersold, and Jean-Michel Bruel. 2022. RQCODE – Towards Object-Oriented Requirements in the Software Security Domain. In *2022 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. 2–6. https://doi.org/10.1109/ICSTW55395.2022.00015 ISSN: 2159-4848.
[13] Rick Rabiser, Sam Guinea, Michael Vierhauser, Luciano Baresi, and Paul Grünbacher. 2017. A comparison framework for runtime monitoring approaches. *Journal of Systems and Software* 125 (2017), 309–321.
[14] Kostadin Rajkovic and Eduard Enoiu. 2022. Nalabs: Detecting bad smells in natural language requirements and test specifications. *arXiv preprint arXiv:2202.05641* (2022).
[15] Andrey Sadovykh, Gunnar Widforss, Dragos Truscan, Eduard Paul Enoiu, Wissam Mallouli, Rosa Iglesias, Alessandra Bagnto, and Olga Hendel. 2021. VeriDevOps: Automated Protection and Prevention to Meet Security Requirements in DevOps. In *2021 Design, Automation Test in Europe Conference Exhibition (DATE)*. 1330–1333. https://doi.org/10.23919/DATE51398.2021.9474185 ISSN: 1558-1101.
[16] Ina Schieferdecker, Juergen Grossmann, and Martin Schneider. 2012. Model-based security testing. *arXiv preprint arXiv:1202.6118* (2012).
[17] Gaadha Sudheerbabu, Tanwir Ahmad, Filip Sebek, Dragos Truscan, Jüri Vain, and Ivan Porres. 2022. A Two-phase Metamorphic Approach for Testing Industrial Control Systems. In *Proceedings of IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2022*. IEEE, Stuttgart, Germany. https://doi.org/NA