Agent-driven security as code for NIS2 compliance in regulated cloud environments*

Ivan Opirskyy[1,†], Oleksandr Vakhula[1,*,†], Pavlo Vorobets[1,†], and Valentyn Fedorchenko[1,†]

[1] *Lviv Polytechnik National University, 12 Stepan Bandera str., 79013 Lviv, Ukraine*

Abstract

Modern regulations such as the EU NIS2 Directive demand a dynamic, automated approach to cybersecurity compliance. This paper proposes an agent-driven Security-as-Code (SaC) methodology to translate NIS2 requirements into enforceable technical controls. In our approach, intelligent agents powered by Large Language Models (LLMs) generate policy-as-code (e.g., OPA Rego policies) from natural-language regulations, guided by domain knowledge bases. We integrate this methodology into a cloud DevSecOps workflow using AWS infrastructure, Terraform (Infrastructure as Code), GitHub Actions CI/CD pipelines, and Open Policy Agent for policy enforcement. A case study on NIS2 Article 21 (risk management measures) demonstrates how high-level legal mandates can be automatically converted into concrete cloud security policies -such as audit logging and network isolation rules -which are enforced in CI/CD to prevent non-compliant deployments. The results show that our agent-driven SaC approach achieves faster policy implementation and improved continuous compliance compared to static, manual methods. We discuss the system's scalability and adaptability for regulated cloud environments, its advantages in maintaining NIS2 compliance, and current limitations (e.g., LLM prompt quality and coverage). The study concludes that an AI-agent paired with Security-as-Code is a promising avenue for proactive NIS2 compliance, and we outline future work on extending this approach to other standards and automated remediation.

Keywords

Security as Code, Policy as Code, Compliance as Code, DevSecOps, cloud security, Infrastructure as Code, security gates, LLM-based agents, NIS2 Directive, regulatory requirements, configuration drift detection, supply chain security.

1. Introduction

The recently adopted NIS2 Directive (Directive (EU) 2022/2555) raises the cybersecurity bar for essential and important entities across the EU, especially those in cloud and critical infrastructure domains. NIS2 significantly expands the scope and stringency of security requirements, mandating measures such as risk analysis policies, incident handling, business continuity planning, supply chain security, secure development practices, and strong access controls. These comprehensive obligations increase the complexity for organizations to maintain continuous security. Non-compliance carries serious consequences — fines up to 10 million euro or 2% of global turnover, personal liability for management, and even service suspension. In this high-stakes environment, effectively and provably implementing NIS2 requirements in cloud infrastructures is imperative.

Translating NIS2's legal mandates into operational practice is challenging. Many organizations struggle to interpret the directive's text into concrete technical controls and processes. Traditional approaches often rely on static security policies and periodic manual audits, which are misaligned with today's rapidly changing cloud and DevOps environments. Static document-based policies can quickly become outdated as cloud infrastructure evolves, and infrequent audits may fail to catch non-compliance in real time. In essence, ensuring that cloud deployments consistently meet NIS2 requirements calls for a dynamic, automated solution integrated into the development lifecycle.

Security-as-Code (SaC) has emerged as a key paradigm to address these challenges. SaC means defining, managing, and enforcing security controls through code and automation rather than manual processes. By integrating security requirements directly into Infrastructure-as-Code

* *CSDP'2026: Cyber Security and Data Protection, May 7, 2026, Lviv, Ukraine*

* Corresponding author.

† These authors contributed equally.

ivan.r.opirskyi@lpnu.ua (I. Opirskyi); oleksandr.p.vakhula@lpnu.ua (O. Vakhula); pavlo.a.vorobets@lpnu.ua (P. Vorobets); valentyn.o.fedorchenko@lpnu.ua (V. Fedorchenko)

0000-0002-8461-8996 (I. Opirskyy); 0009-0008-5367-3344 (O. Vakhula); 0009-0007-3870-829X (P. Vorobets); 0009-0008-0070-1479 (V. Fedorchenko)

templates, configuration scripts, and CI/CD pipelines, organizations can achieve consistent, repeatable enforcement of controls across all stages of development and operations. This approach "shifts left" on security, embedding compliance checks into the cloud deployment lifecycle and reducing human error. Compliance checks become built-in gates in the CI/CD process, rather than afterthoughts. Prior work describes SaC as a "proactive, automated, and integrated security ecosystem" for cloud environments [1]. It allows well-defined controls (access rules, encryption, logging, etc.) to be enforced via code across development, testing, and production, yielding consistent security and compliance outcomes. SaC has been recognized as one of the most effective methods for managing cloud security in the face of evolving threats and requirements.

Despite SaC's promise, implementing it at scale for regulatory compliance remains non-trivial. A recent industry survey [2] found that 94% of organizations consider policy-as-code "vital" for preventative security and compliance at scale, yet many report friction in actualizing it. Key hurdles include the complexity of policy languages (like OPA's Rego) and the effort to keep policies updated across large, dynamic cloud estates. These challenges are amplified when dealing with high-level legal directives like NIS2 that require interpretation and mapping to technical measures. Indeed, industry analysis predicts that through 2025, 99% of cloud security failures will result from customer misconfigurations rather than cloud provider faults [3]. This underscores the need for robust, automated policy enforcement to prevent configuration errors. Early compliance efforts for NIS2 often involve mapping requirements to familiar frameworks (ISO 27001, NIST CSF) to leverage existing controls, but many organizations still find it difficult to implement the new obligations in practice. Traditional compliance approaches (checklists, documentation, periodic audits) may not suffice for NIS2's demand for continuous risk management and rapid incident response. Pan-European assessments also reveal fragmentation and challenges in NIS2 implementation across member states [4]. Clearly, more automated and intelligent solutions are needed to bridge the gap between regulatory requirements and cloud security operations.

In this context, our work introduces an agent-driven Security-as-Code approach for NIS2 compliance. We leverage intelligent software agents (powered by advanced LLMs) to parse NIS2 obligations and automatically generate corresponding cloud security policies in code. This approach aims to provide the agility and responsiveness that manual methods lack: unlike static policies, the AI agents can adapt to changes in the environment or standards by regenerating or updating policies as needed. By using AI to interpret compliance language and produce "policy-as-code" artifacts, we bridge the gap between legal requirements and technical enforcement. In essence, NIS2 requirements are not just written in a policy document -they are codified and continuously enforced in the cloud infrastructure.

2. Related work

Because NIS2 is a relatively new directive (adopted in late 2022), practical guidance on its implementation is still evolving. Reports from the European Union Agency for Cybersecurity (ENISA) emphasize translating NIS2's legal requirements into operational measures [5]. For example, Article 21 of NIS2 mandates a broad set of cybersecurity risk management measures -from policies on risk analysis to incident handling and supply chain security -that organizations must implement. Early guidance often involves aligning NIS2 requirements with existing security frameworks (like ISO 27001 or NIST CSF) to map high-level obligations to known controls. However, many organizations have reported difficulty in understanding and implementing the new obligations, especially given the directive's breadth. There is consensus that purely paper-based compliance (checklists and periodic audits) will not be sufficient for NIS2's continuous compliance mandate. Industry commentators stress the essential role of automation in meeting NIS2 compliance [6-7]. Recent whitepapers also explore how AI and automation can assist organizations in meeting NIS2 requirements [8]. A 2025 study by the European Cyber Security Organisation

identifies significant fragmentation and challenges in NIS2 implementation across EU countries, underscoring the need for more unified, efficient approaches [9].

Security as Code and Policy as Code: The concept of defining security controls as code has gained traction in cloud security. Several recent works have applied SaC to various domains. For instance, one study showed how SaC can fulfill the updated ISO/IEC 27001:2022 security controls by codifying configuration management policies and integrating them into DevSecOps pipelines [9]. Another demonstrated a SaC approach for cloud-native applications on Kubernetes clusters, embedding security policies into Kubernetes configurations [10]. These efforts illustrate that expressing compliance requirements as code can ensure consistent enforcement in diverse environments. Research also highlights Policy-as-Code (PaC) as a means to achieve continuous compliance with standards like GDPR and PCI DSS through automation. By embedding policies (e.g., RBAC/ABAC rules) directly into CI/CD pipelines, organizations can enforce compliance continuously and uniformly. This approach not only prevents drift from security baselines but also greatly improves auditability and evidence collection [11]. However, adopting PaC at scale is challenging in practice -a Styra survey found that while many organizations recognize PaC as vital, nearly half struggle with tooling and integration, though tools like OPA/Gatekeeper are increasingly used [2]. A systematic literature review on DevSecOps compliance found relatively few studies to date, calling for more research into automated compliance integration and metrics to evaluate it [12].

AI for Policy Generation: There is emerging research on applying AI, particularly large language models, to automate security policy creation. Fernandez Saura et al. (2025) present a framework using contemporary LLMs to automate security policy compliance, focusing on generating attack mitigation policies [13]. They leverage in-context learning and retrieval-augmented generation (RAG) to break down high-level policies into actionable tasks, yielding promising results in automatically producing enforcement rules. Similarly, Romeo et al. (2025) introduce ARPaCCino, an agent-based system that employs LLMs with RAG to convert natural language policy descriptions into Rego code and validate them against Infrastructure-as-Code (Terraform) configurations [14]. Their system can iteratively correct infrastructure configurations to comply with the desired policies, effectively "self-healing" non-compliant settings. These AI-driven approaches are at the cutting edge of policy automation, suggesting that LLMs can understand regulatory or policy language and assist in generating enforcement code. Our work aligns with this trend by using LLM agents to interpret NIS2 requirements and produce Rego policies. We extend the idea with a full integration into a CI/CD pipeline for continuous enforcement. Notably, our approach was inspired by prior work in multi-agent systems for SaC: researchers integrated LLMs as part of an AI-driven SaC pipeline and reported faster policy implementation and better compliance adherence compared to manual policy writing [15]. Likewise, the concept of "Intelligent Compliance-as-Code" frameworks has been proposed, combining policy-as-code with machine learning to automate enforcement of standards (e.g., GDPR, HIPAA). Such frameworks have demonstrated significant improvements -one report cited a 65% reduction in compliance-related deployment delays and 98.5% audit coverage through automated checks [16]. These results reinforce that AI and automation can dramatically streamline compliance in DevSecOps environments.

Incident Response and Secret Management: NIS2 places heavy emphasis on incident handling and mitigation. Automation in incident response is a related area where SaC principles apply. Recent research has explored automated security incident management in public cloud environments using SOAR (Security Orchestration, Automation, and Response) tools. By integrating cloud APIs with automated playbooks, such approaches can drastically reduce response times and human error in cloud incident handling. Furthermore, cybercrimes investigation techniques utilizing honeypots in cloud environments [31] provide additional capabilities for threat intelligence gathering and post-incident analysis. Our work focuses on preventive controls, but

such automated response capabilities [17] complement the need for holistic compliance (which includes swift incident response as mandated by NIS2). Additionally, secure software development practices under NIS2 involve proper handling of secrets and credentials. Studies have highlighted the risks of uncontrolled storage of secrets in source code repositories. Exposed API keys or passwords can lead to severe breaches, so NIS2's requirements for secure development and supply chain security implicitly include managing secrets properly. Our policy generation approach, for example, could incorporate checks to ensure no hard-coded secrets or that all secrets are stored in vaults — aligning with recommendations from prior work on secret management [18].

In summary, our contribution builds upon these strands of related work by uniting Security-as-Code enforcement with AI-driven policy generation, specifically in the context of NIS2 compliance. To our knowledge, this is one of the first works to tackle automated NIS2 compliance using an LLM-based agent. It addresses the gap between high-level legal requirements and low-level cloud configurations by automatically generating and enforcing the latter. The next sections describe our proposed methodology and architecture in detail, followed by the case study and results that illustrate the effectiveness of this approach.

3. Proposed methodology

To address the problem of translating regulatory requirements into actionable controls, we propose an agent-driven Security-as-Code methodology that automatically generates and enforces cloud security policies derived from NIS2. The approach comprises several key components working in sequence, forming an adaptive closed-loop system:

- Regulatory Knowledge Base: We first curate a knowledge base of security controls and best practices relevant to NIS2 requirements. This includes mappings from NIS2 articles to specific controls (e.g., mapping Article 21 clauses to ISO 27001 controls, CIS Benchmarks, etc.), as well as examples of policies. This knowledge base provides context to guide the policy generation.
- AI Agent (LLM with Prompting): At the heart of the methodology is an AI agent powered by a Large Language Model. We prompt the LLM with a structured input that provides: (a) the regulatory requirement text (e.g., a clause from NIS2 Article 21), (b) relevant context from the knowledge base (such as corresponding ISO controls or cloud security guidelines), and (c) information about the target environment (e.g., a list of cloud resources from the Infrastructure-as-Code). The agent's task is to output a set of security policy rules (as code) that enforce the given requirement in the context of the environment. We craft the prompt to include examples of input-output pairs for clarity (for instance, an example where the requirement "All S3 buckets must be encrypted" yields a snippet of Rego code checking S3 encryption). This guides the LLM to produce syntactically correct and contextually relevant policy code. If the LLM output is not directly executable or misses details, a developer can perform minor tweaks -but overall, the agent dramatically accelerates the policy creation.
- Policy Generation (Policy-as-Code Artifacts): The output from the LLM agent is parsed into policy-as-code artifacts. In our implementation, these are Open Policy Agent (OPA) Rego rules corresponding to NIS2 controls. Each policy rule is tagged with metadata linking it to the specific NIS2 requirement it addresses (for traceability). For example, a rule enforcing MFA on all user accounts would be tagged to Article 21(i) on access control. This tagging aids transparency and auditing, as one can trace each compliance requirement to the exact code enforcing it. The set of generated policies covers various domains: access control, logging, network security, backup, encryption, etc., aligned to NIS2 clauses. Importantly, this generation step can be repeated or updated whenever requirements change or the environment evolves. The agent can be run periodically or triggered by events (e.g., a new NIS2 guideline or a major infrastructure change) to reassess and update the policy set. This

forms a feedback loop enabling the system to remain up-to-date to a "living" compliance policy that adapts over time.

- Policy Deployment and Enforcement: Once generated, the policies are integrated into the cloud environment's enforcement mechanisms. We adopt a dual approach to enforcement: preventive enforcement in the CI/CD pipeline and, optionally, detective enforcement in runtime. Preventive enforcement is achieved by integrating the policies as tests in the deployment pipeline (using tools like Conftest with OPA). Every time infrastructure changes are proposed (e. g., via a Terraform plan in a pull request), the policies are evaluated; any violation will fail the pipeline and thus block deployment. Detective enforcement can be achieved by deploying OPA policies in the runtime environment (e. g., using OPA Gatekeeper on Kubernetes or AWS Config rules) to continuously monitor and alert on drift from the approved configurations. In our methodology, we focus primarily on the preventive aspect (CI/CD gate), as it ensures compliance issues are caught before reaching production, aligning with the idea of shifting compliance left.
- Continuous Feedback Loop: The methodology supports continuous improvement through feedback. Pipeline results (pass/fail and the specifics of violations) are fed back to both the development team and the knowledge base. For instance, if the agent consistently produces a policy that is too lax or too strict, maintainers can adjust the knowledge base or prompt. Similarly, real-world incidents or new threat intelligence can inform the agent to generate new policies or tighten existing ones. This adaptive loop means the compliance enforcement evolves with both the regulatory landscape and the threat landscape. It reflects a move from static compliance checklists to an agile compliance-as-code practice.

This methodology is design-oriented in this paper. In the next section, we translate this design into a concrete system architecture and implementation details, then evaluate it with a case study. The methodology provides a blueprint that, in principle, could be applied to any compliance framework beyond NIS2 by changing the regulatory input and knowledge context (for example, feeding SOC 2 criteria or GDPR articles to the agent would yield relevant policies, as suggested by prior work [19]). The combination of LLMs with SaC is broadly applicable, it essentially creates a generalizable compliance engine where governance rules in natural language are automatically converted into technical guardrails in the infrastructure.

4. Architecture and implementation

To validate the proposed methodology, we developed a prototype architecture on a regulated cloud scenario using AWS as the environment, Terraform for Infrastructure-as-Code, GitHub Actions for CI/CD, and OPA for policy enforcement. Figure 1 illustrates the architecture (conceptually), and the system consists of the following components:

- Cloud Environment (AWS): We chose Amazon Web Services as the cloud provider for our case study due to its widespread use and rich security features. The environment includes common services: EC2 (virtual machines), S3 (storage buckets), IAM (Identity and Access Management), RDS (managed databases), and VPC (virtual networking). These correspond to areas that NIS2 would consider in scope (ensuring secure configuration of compute instances, proper network segmentation, data protection on storage, etc.). We define and manage this environment using Terraform templates, which serve as the single source of truth for infrastructure configuration. By using IaC, we can apply compliance checks before changes go live.
- Infrastructure as Code (Terraform): All cloud resources and their configurations are expressed in Terraform. When a developer proposes a change (e.g., adding a new S3 bucket or modifying a security group), a Terraform plan is generated in the CI pipeline. This plan (in JSON form) represents the proposed changes to the infrastructure. It becomes the input

for our policy checks. Using IaC in this way is crucial because it allows compliance verification at the planning stage of deployment. In our setup, the Terraform plan file is automatically passed to the policy evaluation step in the pipeline, so any misconfiguration can be caught early. The Terraform state and plan also give us a comprehensive view of the infrastructure, which the LLM agent can use to contextualize its policy generation (for example, knowing which resources exist helps tailor which policies are needed). Recent research has demonstrated the efficiency of centralized configuration repositories for secure cloud service infrastructure management [32], further validating the importance of IaC-driven approaches.

- Policy Engine (Open Policy Agent): We deploy Open Policy Agent in two forms: (1) as a standalone policy evaluator in CI (via the Conftest tool), and (2) as Gatekeeper in a Kubernetes cluster for runtime enforcement (if the deployment involved Kubernetes). For our AWS/Terraform scenario, the primary enforcement is in CI. Each policy generated by the agent is written in Rego (OPA's policy language) and saved as a policy file. In the CI pipeline, we run Conftest against the Terraform plan using these Rego policies. Additionally, we installed OPA Gatekeeper on a test Kubernetes cluster to demonstrate how the same policies could enforce rules in a live cluster (though our case study focus is on AWS IaC). A few examples of implemented Rego policies include:
  - Access Control Policy: Ensure no IAM user or role is created without MFA, and no IAM policies grant wildcard ("*") privileges. This aligns with NIS2's requirements for strict access control and least privilege [Article 21(i)].
  - Logging Policy: Ensure all S3 buckets have access logging enabled, and AWS CloudTrail is capturing management events in all regions. This supports incident handling and audit requirements [Article 21(b),(f)] by guaranteeing audit logs are in place.
  - Network Isolation Policy: Ensure that security groups do not allow wide-open access (e.g., no inbound 0.0.0.0/0 on critical ports) and that sensitive resources reside in private subnets. This aligns with maintaining "information system security policies" and secure network architecture [Article 21(a),(e)].
  - Backup and Resilience Policy: Check that RDS databases have backups enabled (backup retention > 0 days) and, if possible, multi-AZ deployments. This addresses business continuity measures [Article 21(c)] requiring data backup and disaster recovery capabilities.
  - Supply Chain Security Policy: (This is less directly enforceable via Terraform.) We implemented a basic check that all Terraform modules or container images come from approved sources (a simple allow-list), as a nod to supply chain security [Article 21(d)]. While supply chain security also involves processes beyond code (vendor assessments, etc.), this policy helps enforce use of trusted components in infrastructure code.

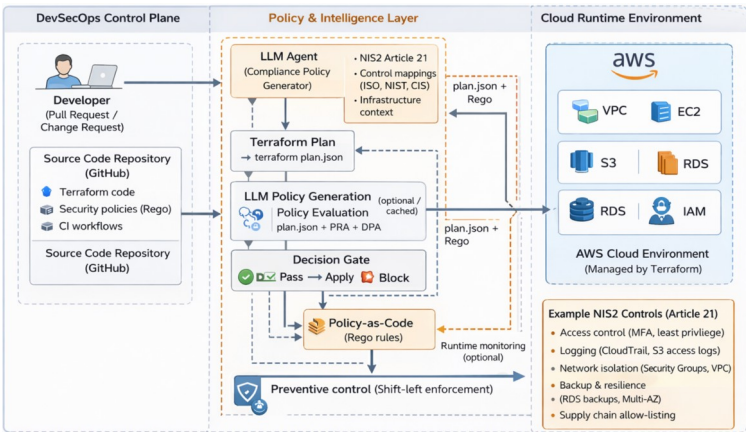

Figure 1: Agent-driven Security as Code architecture for continuous NIS2 compliance in regulated cloud environments.

Each policy file includes metadata linking it to the relevant NIS2 article clause, which is useful for compliance reporting. The novelty of our approach lies in how these Rego policies are generated automatically from high-level requirements using the LLM agent, rather than written manually. We stand on cutting-edge research at this intersection of AI and policy (e. g., LLM-generated policies similar to [13-14]) and integrate it into an end-to-end enforcement pipeline.

- LLM Agent (Policy Generator): We implemented the LLM-based agent using the LangChain framework to manage prompts and model interaction. The agent is containerized (Docker) for easy deployment in the CI pipeline. We experimented with GPT-4 (via OpenAI API) and an open-source LLM (running locally) as the backend. For production use, the choice might depend on privacy and cost constraints (a self-hosted model avoids sending data to an external API). The agent's system prompt defines its role, e. g.: "You are a CompliancePolicyAgent that converts NIS2 requirements into Rego policies using provided context." The dynamic input to the agent includes the exact text of the NIS2 requirement (e. g., Article 21 clause X), relevant control mappings from our knowledge base (e. g., ISO controls related to clause X), and a summary of the current Terraform resources (so the agent knows what kinds of resources to generate policies for). The agent then outputs policy code (Rego rules) as text. We found in testing that GPT-4 was generally capable of producing correct Rego for straightforward requirements (like "enable encryption on S3"). For more abstract requirements (e. g., "ensure secure development practices"), the model sometimes returned high-level guidelines rather than code -highlighting a limitation that not every compliance requirement translates directly into technical rules (we discuss this in the Discussion section). To handle this, we constrained the agent to focus on those parts of NIS2 that have clear technical enforcement paths, and to output a note for purely procedural requirements (e.g., training) that cannot be enforced via code.
- CI/CD Pipeline (GitHub Actions): The CI/CD pipeline ties everything together. Every time changes are pushed to the infrastructure code repository (or a pull request is opened), the pipeline runs. The key stages in our pipeline are:
  1. Terraform Plan: Generate the execution plan (without applying it).
  2. Run LLM Agent: Provide the relevant NIS2 articles (in our case, Article 21 and its sub-clauses) and the Terraform plan context to the agent. The agent either generates new Rego policies or fetches existing ones from a cache. (Optimization: Since NIS2 requirements are relatively static, we pre-generated the core policies for Article 21 and reused them for each run to save time. In a fully dynamic scenario, one could cache the LLM outputs or trigger regeneration only when requirements change significantly) [26].
  3. Policy Check (Conftest): Use Conftest to evaluate the Terraform plan against the set of Rego policies. This yields pass/fail results and messages for each policy.

4. Results and Gate: If all policy checks pass, the pipeline proceeds to apply the Terraform changes (or merge the code). If any check fails, the pipeline stops and reports the compliance violations. For example, a failure might say: "NIS2-21(b) Incident Handling: S3 bucket data-logs does not have access logging enabled (required for audit logging)." The onus is then on the developer to fix the issue (e.g., enable logging on that S3 bucket in the code) and recommit, at which point the pipeline will re-run.

After deployment, we also included a monitoring step for drift detection: certain AWS Config rules and CloudTrail alerts were set to catch changes made outside of Terraform. If, for instance, someone manually turned off an encryption setting in the AWS console, it would trigger an alert. While this is outside the main CI flow, it adds an additional safety net and could be integrated by feeding those alerts back into the agent or pipeline for continuous compliance. Essentially, we aimed to cover both preventive controls (CI/CD gates) and detective controls (runtime monitoring), as recommended in best practices [20].

Our implementation heavily leverages existing DevOps tools and infrastructure, showing that the approach can be adopted without reinventing the wheel: Terraform provides the IaC interface, OPA/Conftest provides the policy enforcement mechanism, and the LLM agent injects intelligence into policy creation. We also ensured that all components are modular. For example, one could swap AWS/Terraform for Azure/ARM or Kubernetes manifests, and the core approach would remain the same (only the prompt context for the agent and some policy specifics would change).

Overall, the architecture demonstrates how compliance-as-code can be achieved in practice. It places an automated compliance gate in the CI/CD process to catch any violation of NIS2-driven policies before deployment (similar ideas have been discussed by others [2]). The entire system is a concrete instantiation of using SaC plus AI to achieve continuous compliance. Next, we describe a case study using this system and the outcomes observed.

5. Case study and results

To evaluate the effectiveness of the proposed agent-driven SaC approach, we conducted a case study focusing on Article 21 of NIS2. Article 21 enumerates core cybersecurity risk-management measures that organizations must implement, covering multiple facets of security (policies on risk analysis, incident handling, business continuity, supply chain security, secure development, access control, etc.). It is an ideal candidate for a case study because it spans technical controls that can be implemented and checked in a cloud environment.

Scenario Setup: We simulate an organization deploying an AWS cloud application (e. g., a web service handling sensitive data) that falls under NIS2 obligations (for instance, a digital service provider in scope of the directive). The infrastructure, defined via Terraform, includes a typical setup with multiple components:

- Network: A VPC with both public and private subnets.
- Compute: EC2 instances in private subnets (for application servers) and a bastion host in a public subnet for administrative access.
- Database: An Amazon RDS database instance for persistent data.
- Storage: S3 buckets for data storage and log archival.
- Identity: IAM users and roles for administrators and application services.
- Logging: AWS CloudTrail enabled to record API calls (for auditing).

We intentionally introduced a few misconfigurations and omissions in this setup to represent common security gaps:

- An S3 bucket lacks server-side encryption and has no access logging.
- The RDS database has automated backups disabled (retention period set to 0).
- A security group allows inbound traffic from anywhere (0.0.0.0/0) to an administrative port (SSH on port 22).

- The IAM account has no password policy set (meaning password complexity and rotation requirements are not enforced).
- CloudTrail is only logging activity in one region, while the organization uses multiple AWS regions (leaving a visibility gap).

These issues correspond to several Article 21 requirements:

- Lack of encryption and logging touches on incident handling and data security (Article 21(b) and 21(f) require audit logging and event handling).
- Disabled backups affect business continuity (Article 21(c) requires disaster recovery measures).
- Open network access violates information system security policies (Article 21(a) calls for securing network and systems).
- No password policy undermines access control (Article 21(i) requires access controls and likely implies having credential policies).
- Partial logging (single-region) means risk analysis and monitoring are incomplete, reducing the effectiveness of security measures (Article 21(e) and 21(h) emphasize security of processes and cryptography, indirectly requiring comprehensive monitoring).

Before using our system, these misconfigurations would make the deployment non-compliant with NIS2. In a traditional setup, they might go unnoticed until an audit or, worse, be exploited.

Policy Generation: We configured the LLM agent with Article 21's text and our knowledge base context, then executed it to generate a suite of Rego policies for the scenario. The agent produced policies mapped to the key clauses of Article 21, for example:

- Audit Logging Policy (maps to Article 21(b) incident handling and 21(f) effectiveness of measures): Ensure CloudTrail is enabled for all AWS regions and that every S3 bucket has access logging turned on. The agent recommended this based on the incident handling clause, reasoning that comprehensive logging is needed to handle and assess incidents.
- Network Security Policy (maps to Article 21(a) information security policies): Flag any security group rule that allows wide-open access (especially on critical ports like SSH or database ports). This enforces network isolation and least exposure of services.
- Backup and Resilience Policy (maps to Article 21(c) continuity): Require that RDS instances have a backup retention period > 0 (i.e., backups enabled) and, if possible, that they use multi-AZ deployment for high availability. Also suggest that EC2 volumes should have snapshots or a backup plan. These ensure data backup and resilience.
- IAM Policy (maps to Article 21(i) access control and relates to human resources security): Require that a strong account password policy is in place (e.g., minimum length, complexity, rotation), that IAM users have MFA enabled, and that no long-unused credentials remain active. Interestingly, the agent added a check for unused credentials older than 90 days -this was not explicitly in NIS2, but it aligns with best practices and the spirit of "basic cyberhygiene" in staff credential management.
- Encryption Policy (maps to Article 21(e) secure development and 21(h) cryptography): Ensure that all data at rest is encrypted -e. g., S3 buckets have encryption enabled and RDS storage is encrypted. Also ensure data in transit is protected, for instance that any application load balancer has TLS enabled. This addresses cryptographic protection requirements.
- Supply Chain Policy (maps to Article 21(d) supply chain security): This was the most challenging for the agent. It proposed a rule that "all third-party Terraform modules must be pinned to specific versions and from approved sources." We accepted this as a partial technical measure (since supply chain security also involves vendor assessments beyond code). It at least enforces that no unvetted or latest-version modules are pulled, reducing risk of pulling malicious code.

In total, the agent generated 8 Rego rules covering these areas. We reviewed them and found most to be on point. Two policies needed minor adjustments: the CloudTrail policy originally did not consider multi-region logging (we tweaked it to ensure logging is global), and the MFA rule needed to account for both IAM user accounts and the root account. These tweaks were small and took only minutes, confirming that the heavy lifting of policy creation was handled by the AI agent.

CI/CD Enforcement: With the policies ready, we ran the GitHub Actions pipeline on the Terraform code containing the misconfigurations. As expected, the pipeline failed, flagging multiple compliance violations:

- Logging Policy Violation: It reported that CloudTrail was only enabled in one region (us-east-1) and not all regions, citing NIS2 Article 21(b) for incident logging. It also flagged a specific S3 bucket missing access logging. These correspond to our intentional gaps.
- Backup Policy Violation: It caught the RDS instance with backup_retention_period = 0, citing Article 21(c) on continuity. This indicated no backups were configured, which is non-compliant.
- Network Policy Violation: It flagged the security group rule allowing 0.0.0.0/0 access on port 22, citing Article 21(a), and recommended restricting it (e.g., to a specific IP range).
- IAM Policy Violation: It noted that no IAM password policy was present, citing Article 21(i) (access control). It also noted that two IAM users did not have MFA enabled (we had created test users to simulate this issue).
- Encryption Policy Violation: It flagged the unencrypted S3 bucket and the RDS instance not having storage encryption turned on, citing Article 21(e) and 21(h) requirements for encryption.

Each violation message was prepended with a code like "NIS2-21(x)" linking it to the specific clause, which is helpful for compliance traceability. This output provided a clear to-do list for the engineers to remediate. We proceeded to fix each issue in the Terraform code:

- Enabled CloudTrail for all regions (a one-line change in the Terraform CloudTrail resource).
- Enabled access logging on the S3 bucket (adding a logging configuration).
- Set the RDS backup retention to a non-zero value (e. g., 7 days).
- Restricted the open security group rule to the corporate IP range (simulating that only our office IP can SSH, or alternatively removing the rule).
- Added an IAM account password policy resource with strong settings (minimum length, require symbols, etc.).
- Marked the test IAM users as requiring MFA or removed those test accounts.
- Enabled default encryption on the S3 bucket and enabled storage encryption for RDS (in Terraform these are simple boolean flags).

After these fixes, we re-ran the pipeline. This time, all compliance checks passed, and the infrastructure changes were allowed to deploy. The system effectively prevented a non-compliant infrastructure configuration from being applied, guiding us to a compliant state. This demonstrates the value of shifting compliance left into the CI process: rather than discovering these issues later (or suffering a breach/fine), they were caught and resolved immediately during development.

Results Analysis: The case study yields several important observations:

- Automated Compliance Enforcement: We achieved end-to-end automation from interpreting a requirement to enforcing it. The final cloud environment state was fully in line with NIS2 Article 21 measures -logging enabled, backups in place, strict access controls, encryption enforced -all without relying on a manual compliance checklist. Compliance was enforced by code. This fulfills the directive's intent that such measures "shall be taken" continuously, and it provides evidence for auditors (the CI pipeline logs

and version-controlled policies serve as audit evidence of controls in place). Essentially, we demonstrated that code-driven compliance is feasible and effective.

- Agent Efficacy: The LLM agent was largely successful in generating meaningful policies. Out of 8 rules, 6 were immediately useful and correct; 2 needed only minor fixes. The speedup is significant: writing those policies from scratch could have taken a security engineer days of work (especially if not deeply familiar with Rego syntax), whereas the agent produced a draft set in minutes. In some cases, the agent even covered considerations we might have overlooked -for example, adding a check on stale IAM credentials (which, while not explicitly in NIS2, is a best practice). This shows that a well-trained model can inject broader security knowledge into the compliance process, an added benefit of using AI.
- Continuous Compliance in Action: After deploying the compliant setup, we simulated a configuration drift scenario to test continuous monitoring. We manually turned off encryption on one S3 bucket via the AWS console (to mimic an admin making an out-of-band change). Our detective controls caught this: an AWS Config rule triggered an alert for the unencrypted bucket. We also ran our OPA policies against the live environment (using opa eval on the current state) and it flagged the same issue. This indicates that our approach can extend to ongoing monitoring: one could schedule daily or weekly scans of the environment against the policies, or use tools like Cloud Custodian in parallel, to ensure nothing has silently fallen out of compliance. In a mature setup, the same set of policy rules could be used both in preventive mode (CI enforcement) and detective mode (continuous audit), maximizing reuse and consistency.
- Scalability: Our case study was on a small scale (~100 cloud resources), but the generated policies were generic and should scale to larger environments. OPA's policy evaluation is efficient and can handle hundreds or thousands of resource checks quickly. The primary consideration for scaling is maintainability of the policy set as things change. However, because our agent can re-run on demand, if the organization adds new resource types or if NIS2 evolves, new policies can be generated to cover those. For example, if the organization later adopts Kubernetes, we could feed the agent with NIS2 requirements plus Kubernetes context to get additional policies (like checking that pods are not running as root, etc., if relevant to NIS2). Thus, the approach appears scalable both in technical enforcement and in ability to evolve with the environment.

We also performed a qualitative comparison against a traditional compliance approach for context. Manually, an engineer would have to read through NIS2 Article 21, interpret each requirement, map it to AWS/Terraform controls, then inspect the Terraform configurations or cloud console to verify compliance. Doing that thoroughly could take many hours and might miss subtle points (for instance, noticing CloudTrail was only regional). It's also just a point-in-time check -after the audit, any new change could break compliance unbeknownst to the team until the next audit. In contrast, with our automated approach, once set up, every change is instantly checked. This closes the window of vulnerability and ensures compliance is not a periodic project but a continuous background process. This observation is in line with industry experiences that emphasize integrating compliance into DevOps to avoid the inefficiencies of separate audit cycles [21-22].

Limitations Observed: We should note that not every aspect of NIS2 Article 21 can be automatically enforced by technical means. Some requirements are procedural or organizational. For example, Article 21(g) requires implementing "basic cyber hygiene practices and cybersecurity training." Such measures involve human processes (training programs, employee awareness) that cannot be enforced through code. Our system cannot train employees or verify they completed training. In these cases, the best an automated system can do is provide reminders or documentation. In fact, our agent output for 21(g) included a comment like "Ensure all staff have

completed security awareness training -cannot be enforced via code -for compliance." We included that note in the compliance report as an informational item. This highlights that our approach covers a subset of compliance (the technical controls) and should be complemented with governance processes for the procedural controls. Similarly, supply chain security (21(d)) extends beyond just using approved modules; it involves vetting suppliers and software components, which is partially outside the scope of an infrastructure-focused tool. We documented these gaps to make clear that our system is meant to augment, not completely replace, a full compliance program. Nonetheless, even just covering the technical controls significantly reduces the compliance burden and risk, while the remaining procedural items can be handled by traditional governance with much less effort.

Overall, the case study confirmed that agent-driven Security-as-Code is a viable and effective approach for NIS2 compliance automation. We were able to automatically enforce multiple NIS2 controls and adapt the cloud deployment accordingly. The "gate fail" mechanism in CI worked as intended -preventing insecure configurations from being deployed and thereby likely preventing incidents or regulatory penalties that could result from those misconfigs. This kind of proactive enforcement is exactly what regulations like NIS2 aim to achieve across organizations, and our results show a practical path to achieve it through automation.

In conclusion, the case study's AWS infrastructure was brought into compliance with NIS2 Article 21 using our system, and it stayed compliant through changes due to the continuous checks. Next, we discuss the broader implications, advantages, and limitations of our approach, and how it compares to alternative methods.

6. Discussion

The development and evaluation of this agent-driven SaC approach for NIS2 compliance yield several insights into its advantages, limitations, and its position relative to other methods.

Advantages and strengths:

- Scalability and Consistency: One of the most significant benefits is scalability. Once the LLM-generated policies are in place, they can be applied consistently across any number of cloud resources and deployments. This addresses the problem of human error and omission in large environments. The codified policies act as a scalable guardian: whether there are 10 cloud resources or 10,000, each is automatically checked against the same standards. This consistency greatly improves auditability as well -every evaluation is performed exactly according to policy. As noted in prior work, such automation ensures uniform policy enforcement across diverse environments and improves audit readiness [11]. Our results align with the conclusion that Policy-as-Code enables continuous compliance and operational efficiency, especially in complex, multi-team cloud setups. For NIS2, which will impact a wide range of organizations and sectors, an automated approach can ensure each entity adheres to a baseline set of controls uniformly, something very hard to achieve with manual processes.
- Adaptability (Agent-Driven Evolution): Using an intelligent agent introduces adaptability that static approaches lack. If NIS2 is amended or expanded in the future (e. g., a new requirement on a novel domain like AI security gets added), we can feed the updated text to the agent and quickly generate new or modified policies without overhauling the whole compliance program. Likewise, as our infrastructure changes or new cloud services are adopted, the agent can be prompted to extend policies to cover those. This is a major improvement over traditional code-based policies that would require manual updates for each change. It reflects a shift towards living policies -policies that evolve in tandem with the regulatory context and the system environment. This concept is analogous to adaptive security in which defenses evolve with the threat landscape; here our compliance controls

evolve with regulatory and infrastructural changes, facilitated by AI. Essentially, our approach offers a degree of future-proofing: the AI agent's broad knowledge can incorporate new standards on the fly, and its output can grow as needed. This adaptability will become increasingly important as regulations like NIS2 are periodically updated and as organizations juggle multiple frameworks (e.g., a company might need to comply with NIS2 and another standard like SOC 2 or GDPR; an adaptive agent could potentially handle multi-framework requirements in one go, as suggested in future work) [27, 28].

- Real-Time Compliance (Continuous Enforcement): By integrating compliance checks into CI/CD, compliance becomes an ongoing, real-time process rather than a periodic audit exercise. This has huge implications for risk reduction -misconfigurations are caught before they can cause an incident or violation. It transforms compliance from something retrospective (finding issues after the fact) to proactive (preventing issues from ever reaching production). As a whitepaper on policy-as-code notes, continuous policy enforcement can drastically reduce the window of exposure and the effort spent on manual audits [9]. In our case, developers get immediate feedback on compliance violations, which also has a training effect: over time, they learn the expected security configurations (since non-compliance breaks their build). This fosters a culture of "security built-in" rather than "security audited later." Moreover, continuous compliance provides up-to-date evidence at any time -you could generate a compliance report on demand from pipeline logs, which is valuable for internal compliance officers and external regulators alike [29].
- Improved Security Posture: A side effect (or rather, an intended outcome) of compliance-as-code is that it directly improves the actual security posture, not just compliance on paper. By implementing NIS2 measures via code, we are hardening the infrastructure -encryption is enabled, logging is comprehensive, access is locked down, etc. So, beyond ticking boxes for auditors, the organization becomes more resilient to attacks. Our case study demonstrated tangible security improvements (e.g., closing an open SSH port, turning on encryption and backups) that reduce incident risk. This addresses NIS2's fundamental goal: to raise the overall level of cybersecurity. We believe this dual benefit (regulatory compliance + real security gains) is a compelling argument for automated compliance approaches. It ensures that meeting compliance requirements isn't just a checkbox exercise, but actually correlates with better protection of systems and data.
- Transparency and Traceability: With policies codified and linked to requirements, it is easier to demonstrate compliance. Auditors or management can inspect the Rego policies to see exactly how each NIS2 requirement is enforced. There's less ambiguity than in narrative documents -for example, a code policy enforcing "S3 buckets must be encrypted" is unambiguous and testable, whereas a written policy might just state "we encrypt data" without clear proof. This transparency can streamline audits. Each failed pipeline check is traceable to a specific requirement clause, which creates an automatic paper trail of compliance status. During an audit or review, one could show: "Here are our controls (as code) for each NIS2 clause, and here are the pipeline logs showing they are enforced on every change." This kind of evidence is much stronger than a periodic report.

Discussion of Related Approaches: It is worth comparing our approach to other emerging compliance solutions:

Traditional Governance, Risk, and Compliance (GRC) tools often focus on documentation and workflow for compliance (managing policies, risk registers, audit findings). Those are important at an organizational level, but they don't enforce technical controls. On the other hand, cloud security posture management (CSPM) tools do automatically scan cloud environments for misconfigurations (some are even NIS2-mapped). CSPMs, however, typically detect issues after deployment and often are read-only advisory tools. Our approach is more preventive and

developer-facing, integrating into the DevOps toolchain. There are also Infrastructure-as-Code static analysis tools (like terraform-linters, Polaris for Kubernetes, etc.), but they usually use fixed rule sets. Our agent-driven method can generate rules dynamically from high-level inputs, which is novel. It's also complementary to those tools -for example, one could integrate custom generated rules with existing rule sets for comprehensive coverage.

Another comparison is with the "Compliance-as-Code" frameworks that some organizations are building, often leveraging OSCAL (an open standard for compliance as code) or products like RegScale. Those frameworks often involve codifying controls in a structured format and mapping them to evidence. Our approach could be seen as an automated way to produce those control definitions and evidence checks. We directly enforce, whereas some compliance-as-code efforts focus on tracking and reporting compliance status. Both approaches could converge our system could feed results into a compliance dashboard.

Challenges and Limitations: despite the advantages, there are challenges:

- Reliance on AI Correctness: The approach assumes the LLM agent produces correct and secure policies. If the LLM misunderstood a requirement or if there is a gap in the knowledge base, it might generate an incomplete policy. Human oversight is still needed when onboarding new requirements to verify the suggested controls make sense. In safety-critical contexts, one might require that a human security engineer reviews all AI-generated policies before they are trusted. Fortunately, once vetted, the policies remain consistent until requirements change. The initial verification is crucial.
- Prompt Maintenance: As both NIS2 and cloud services evolve, the prompts and knowledge base for the LLM will need updates. For example, if AWS introduces a new service that falls under existing requirements (say a new type of storage service), we'd need to add that to the knowledge base or adjust prompts so the agent knows how to handle it. Ongoing maintenance of the AI's context is an overhead to plan for.
- Coverage of Requirements: We saw that some NIS2 requirements are not enforceable in code. This means our system can't give 100% NIS2 compliance in isolation; it covers the technical controls. Organizations must still address governance aspects (like policies for training, incident response process documents, etc.) outside the tool. Our system could integrate with those processes by outputting tasks or reminders (as it did by noting the training requirement), but it won't automate them. In practice, one would use our approach alongside a broader compliance program.
- False Sense of Security: There is a slight risk that reliance on automated checks might make organizations complacent. They might assume "if the pipeline passes, we are secure." While pipeline passing does mean NIS2 controls are in place, it's not a guarantee of total security. New types of misconfigurations or threats beyond the scope of the policies could arise. Regular reviews and updates of the policy set are important to ensure they cover emerging risks (for example, if cryptographic best practices change, the encryption policy should be updated accordingly). Thus, human governance shouldn't disappear; it should become more efficient and data-driven thanks to these tools.
- Integration Effort: Implementing this system requires integrating several tools (CI/CD, OPA, LLM hosting) and writing glue code (for prompt engineering, parsing outputs, etc.). For smaller organizations, this might be non-trivial. However, once set up, it largely runs automatically. In our case study, setting up the initial pipeline and policies took significant effort, but running it thereafter was smooth. The investment is front-loaded. Over time, as such approaches mature, we expect more off-the-shelf solutions or open-source stacks that make it easier (for instance, one can imagine a turnkey "Compliance Agent" product emerging in the market given the interest in this area [22]).

In comparison to alternative methods, our approach offers a unique blend of automation and intelligence. Some organizations may opt for purely manual compliance supplemented by periodic

scanning tools; that route is lower initial effort but higher ongoing risk and labor. Others might develop static policy-as-code by hand; that ensures precision but is labor-intensive and less adaptive. Our agent-driven approach attempts to get the best of both: automate the heavy lifting of policy creation and keep the human in the loop for oversight and fine-tuning.

Looking outward, a few contemporaneous efforts are worth noting. The research community and industry are actively exploring AI for security compliance (as evidenced by [13, 14] and various industry whitepapers [8, 23]). We see our work as part of this emerging trend. Projects like Praetorian in Europe aim to build resilience in critical infrastructure and share incident information, which complements NIS2 enforcement [24]. Other recent studies are examining how NIS2 will influence cybersecurity models and IT governance [24, 25]. All indicate that automation and intelligent tools will be essential to meet the directive's goals at scale [30].

7. Conclusions

In this work, we proposed an agent-driven Security-as-Code approach for achieving compliance with the NIS2 Directive in cloud environments. By leveraging Large Language Models as intelligent agents, we demonstrated how high-level regulatory requirements can be systematically translated into concrete, enforceable security policies embedded directly into cloud infrastructure and CI/CD pipelines. The proposed architecture, integrating AWS resources, Terraform Infrastructure as Code, Open Policy Agent enforcement, and an LLM-based policy generator, provides a practical blueprint for automating and scaling cybersecurity compliance in regulated environments.

The results confirm the feasibility of AI-generated policy-as-code artifacts for regulatory enforcement. The agent successfully produced a comprehensive set of OPA/Rego policies addressing the major technical measures required by NIS2 Article 21, significantly reducing the manual effort and expertise normally required for compliance engineering. Embedding these policies into the CI/CD pipeline enabled continuous and preventive compliance, where violations were detected and remediated at build time rather than through periodic audits. This shift to automated compliance gates aligns with the proactive risk management model mandated by NIS2 and ensures that compliance is maintained as an integral part of the deployment process

Beyond formal compliance, the approach improved the actual security posture of the cloud environment. Controls implemented as code — such as encryption, access control, logging, and network segmentation — directly translated into hardened infrastructure, thereby reducing real cyber risk while satisfying regulatory obligations. The solution also proved scalable and auditable: the same codified controls were uniformly applied across environments, and CI/CD logs combined with version-controlled policies provided built-in audit evidence, enabling clear traceability from regulatory requirement to technical enforcement result [13].

Importantly, the study demonstrates how AI can bridge the long-standing gap between natural-language compliance requirements and technical enforcement mechanisms. The proposed methodology is not limited to NIS2 and can be adapted to other regulatory frameworks, supporting the broader evolution toward Compliance-as-Code and automated governance systems. At the same time, certain limitations remain. The approach does not fully cover non-technical NIS2 controls, such as personnel training or organizational procedures, which still require human-led governance. In addition, the correctness of AI-generated policies depends on the quality of prompts and knowledge bases, making human oversight essential. Maintaining up-to-date regulatory context is therefore a critical requirement for long-term accuracy.

Future work will focus on extending the agent to support multi-framework compliance, enabling unified policy generation for overlapping requirements across NIS2, ISO/IEC 27001, and SOC 2. Another direction is automated remediation, where the system could suggest or apply corrective changes with human approval, further reducing time to compliance. Incorporating feedback from incidents and near-misses could allow policies to adapt dynamically, creating a self-improving

compliance system. Enhancing developer experience through IDE plugins or real-time compliance feedback dashboards would also support adoption, while large-scale evaluations in enterprise environments are needed to quantify efficiency gains and risk reduction. Finally, integrating procedural controls into automated tracking workflows (e. g., training schedules or incident response exercises) would further close the gap between technical and organizational compliance.

Overall, this study provides evidence that Security-as-Code, augmented by AI agents, is a viable and effective path for maintaining compliance in modern cloud environments. The approach transforms compliance from a static checklist into a continuously enforced, automated process embedded in DevOps workflows. For regulations like NIS2, which aim to improve cybersecurity resilience at scale, agent-driven Security-as-Code offers a practical mechanism for aligning regulatory intent with operational reality. As regulatory complexity and cloud dynamism continue to grow, the convergence of compliance, security engineering, and artificial intelligence demonstrated in this work is likely to become a standard practice rather than an exception.

Declaration on Generative AI

While preparing this work, the authors used the AI programs Grammarly Pro to correct text grammar and Strike Plagiarism to search for possible plagiarism. After using this tool, the authors reviewed and edited the content as needed and took full responsibility for the publication's content.

References

[1] O. Vakhula, I. Opirskyy, O. Mykhaylova, Research on Security Challenges in Cloud Environments and Solutions based on the "Security-as-Code" Approach. In: Proc. of CPITS 2023, CEUR Workshop Proc., vol. 3550, pp. 55-69.

[2] J. Bourne, Policy as Code a Strategic Imperative -but Scalability Remains Difficult. CloudComputing-News, Nov 2, 2023.

[3] IBM, Cloud security evolution: Years of progress and challenges. IBM Security Article (citing Gartner prediction on cloud misconfiguration), 2023.

[4] European Cyber Security Organisation, 2024. NIS2 Implementation: Challenges & Priorities. ECSO White Paper.

[5] ENISA, Cybersecurity Roles and Skills for NIS2 -Mapping NIS2 Obligations to ECSF Profiles. Eur. Union Agency Cybersecur. Rep. (via CyberHubs News), 2025.

[6] D. Blackwell, The Essential Role of Automation to Meet NIS2 Compliance. Swimlane Blog, Jan 29, 2024.

[7] ControlMonkey, Automating NIS2 Compliance for DevOps Teams. ControlMonkey Blog, Oct 20, 2025.

[8] ThirdWave Identity, The Role of AI and Automation in NIS2 Compliance. Third Wave Identity White Paper, 2024.

[9] O. Vakhula, Y. Kurii, I. Opirskyy, V. Susukailo, Security-as-Code Concept for Fulfilling, Requirements. In Proc. of CPITS 2024, CEUR Workshop Proc., vol. 3654, pp. 59-72.

[10] O. Vakhula, I. Opirskyy, Research on Security as Code Approach for Cloud-Native Applications based on Kubernetes Clusters. In: Proc. of CSDP 2024, CEUR Workshop Proc., vol. 3800, pp. 58-69.

[11] O. Vakhula, I. Opirskyy, P. Vorobets, O. Bobko, O. Kulinich, Policy-as-Code for Implementing Role-Based and Attribute-Based Access Control in Cloud-Native Systems. In Proc. of CPITS, 2025, CEUR Workshop Proc., vol. 3991, pp. 139-157.

[12] X. Ramaj,et al., Holding on to Compliance While Adopting DevSecOps: A Systematic Literature Review. Electronics, 11(22), 3707, 2022. doi:10.3390/electronics11223707.

[13] P. Fernández Saura, et al., On Automating Security Policies with Contemporary LLMs, In: IEEE SSE 2025.

[14] F. Romeo, ARPaCCino: An Agentic-RAG System for Policy-as-Code Compliance, 2025.
[15] O. Vakhula, et al., AI-driven Security as Code for Software Development using Multi-Agent Systems. In: Proc. of CH&CMiGIN 2025, CEUR Workshop Proc., vol. 4024, pp. 170-185, Kyiv, Ukraine.
[16] T. Edgar, M. Crowley, C. Halbrook, Intelligent Compliance-as-Code Frameworks for DevSecOps. ResearchGate Preprint, 2024.
[17] O. Sapsai, Y. Martseniuk, A. Partyka, O. Harasymchuk, Research on Automated Security Incident Management in Public Cloud Environments. In: Proc. of CSDP 2025, CEUR Workshop Proc., vol. 4042, pp. 226-249), Lviv, Ukraine.
[18] V. Chornii,Y. Martseniuk, A. Partyka, O. Harasymchuk, Information Security Risks Associated with the Uncontrolled Storage of Secrets in Source Code, In: Proc. of CSDP 2025 CEUR Workshop Proc., vol. 4042, pp. 250-271, 2025, Lviv, Ukraine.
[19] Y. Martseniuk, A. Partyka, I. Opirskyy, O. Harasymchuk, Cost Observability as a Security Control in Multi-Cloud Environments Based on SOC 2 Security Standard. Computers & Security, 120, 104771, 2025. doi:10.1016/j.cose.2025.104771.
[20] CypSec, Policy-as-Code Service -Core Features and Compliance Integration. CypSec Security Center (Whitepaper), 2025.
[21] QualySec, NIS2 Directive Requirements & Compliance Guide. QualySec Blog/Guide, 2025.
[22] SentinelOne, AI Security Solutions & Controls -CI/CD Best Practices. SentinelLabs Article, 2025.
[23] Darktrace, NIS2 Compliance: Interpreting “State-of-the-Art” for Organisations. Darktrace Blog, Oct 2024.
[24] M. Veigurs, T. Lasmanis, A. Romanovs, IT Governance in Critical Sectors: Towards the NIS2 Implementation, In: Proc. of the IEEE 65th International Conference on Information Technology and Management Science (ITMS 2024), pp. 1-7), Riga, Latvia: IEEE.
[25] P. Wanecki, R. Jašek, I. Drofová, The Contribution of the European NIS2 Directive to the Design of the Cyber Security Model, In: Proc. of the International Conference on Information and Digital Technologies (IDT 2023), pp. 149-154. Žilina, Slovakia: IEEE.
[26] Y. Zhuravchak, D. Zhuravchak, A. Zhuravchak, I. Opirskyy, Security Regression Visualization in CI/CD Using Trivy and GitHub Actions, In: Proc. of CSDP 2025, CEUR Workshop Proc., vol. 4042, pp. 272-281), Lviv, Ukraine.
[27] Directive (EU) 2022/2555 (NIS2 Directive) – Article 21: Cybersecurity risk-management measures.
[28] Cloud Security Alliance, Compliance as Code is the Future -How to Get Started. CSA Article, 2023.
[29] ISACA, Security-as-Code: A Key Building Block for DevSecOps. ISACA Insights, 2022.
[30] OpsMx, Automating Compliance in DevSecOps to Improve AppSec Posture. OpsMx Blog, 2023.
[31] V. Susukailo, S. Vasylyshyn, I. Opirskyy, V. Buriachok, O. Riabchun, Cybercrimes Investigation via Honeypots in Cloud Environments, In: Proc. of CPITS 2021, CEUR Workshop Proc., vol. 2923, pp. 91-96.
[32] Y. Martseniuk, A. Partyka, O. Harasymchuk, V. Cherevyk, N. Dovzhenko, Research of the Centralized Configuration Repository Efficiency for Secure Cloud Service Infrastructure Management, In: Proc. of CPITS 2025, CEUR Workshop Proc., vol. 3991, pp. 260-274.