

Security-aware centralized network configuration management with automated compliance verification

Oleksandra Yeremenko^{1,*†}, Maksym Trubeko^{1,†} and Roman Savchenko^{1,†}

¹Kharkiv National University of Radio Electronics (NURE), Nauky Ave., 14, Kharkiv, 61166, Ukraine

Abstract

This paper proposes an architecture for centralized management of network device configurations based on the Ansible platform, with support for automated security policy compliance monitoring. The approach combines the principles of Infrastructure as Code, automated configuration deployment, security compliance verification, detection of configuration deviations, and automatic restoration of the network infrastructure to its baseline state. To guarantee compliance with modern information security requirements, a set of security policies was developed and mapped against the requirements of ISO/IEC 27001:2022, NIST SP 800-53 Rev. 5, and CIS Controls v8. The functional implementation of the proposed approach was carried out in a test environment comprising network devices from various manufacturers and centralized management tools. An experimental study was conducted on the processes of deploying configurations, verifying compliance with security policies, detecting configuration deviations, and automatically resolving identified violations. The results confirmed the effectiveness of the proposed architecture and the feasibility of an automated approach for managing network infrastructure. The proposed solution can be used to build centralized network device configuration management systems with built-in mechanisms to monitor and maintain compliance with information security requirements.

Keywords

network automation, configuration management, security compliance, configuration drift detection, automated remediation, Ansible

1. Introduction

Modern network infrastructures are characterized by high complexity, heterogeneous equipment, and a constant increase in the number of configuration changes [1, 2]. In such conditions, guaranteeing the consistency of network device configurations and compliance with security policies becomes one of the key tasks of network administration [3, 4, 5]. Manual configuration management not only increases operational costs but also raises the risk of errors that may lead to security breaches and vulnerabilities in the network infrastructure. The development of network automation approaches and the Infrastructure as Code (IaC) concept creates the conditions for centralized configuration management using automation tools [6, 7, 8]. However, the implementation of automated mechanisms alone does not guarantee that configurations comply with security requirements. The task of integrating processes to deploy configurations, validate compliance with security policies, detect configuration drifts, and automatically restore the correct state of network devices is still relevant.

This work aims to develop and study an architecture for centralized configuration management of network devices with support for automated verification of compliance with security policies. The proposed approach integrates centralized configuration deployment, compliance monitoring with established security policies, detection of configuration drifts, and automated remediation of violations.

To evaluate the effectiveness of the proposed solution, a test environment comprising network devices from various manufacturers was implemented, and an experimental study of the time characteristics of drift detection, compliance verification, and configuration restoration was conducted. The results

CMiGIN'26: Fifth International Workshop on Conflict Management in Global Information Networks, June 26–27, 2026, Krakow, Poland

*Corresponding author.

† These authors contributed equally.

✉ oleksandra.yeremenko@nure.ua (O. Yeremenko); maksym.trubeko@nure.ua (M. Trubeko); roman.savchenko1@nure.ua (R. Savchenko)

ORCID: 0000-0003-3721-8188 (O. Yeremenko); 0009-0009-0340-598X (M. Trubeko); 0009-0005-6693-8733 (R. Savchenko)



© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

prove the feasibility of a centralized approach to managing network device configurations with built-in security controls.

The remainder of this paper is organized as follows:

1. Section 2 provides background information and explains the technology selection process.
2. Section 3 presents the architecture of the proposed Security-Aware Centralized Configuration Management system.
3. Section 4 describes the implementation of the proposed solution and evaluates its effectiveness through experimental validation.
4. Section 5 concludes the paper and discusses future research directions.

2. Background and technology selection

IaC automation platforms are used to implement centralized management of network device configurations; the most common of which are Ansible, SaltStack, Puppet, and Nornir [9, 10, 11, 12, 13]. In the context of this study, the key selection criteria are support for heterogeneous network equipment, agentless architecture, idempotent operations, and the ability to automate compliance checks against security policies.

Puppet and SaltStack systems primarily use an agent-based management model, which requires installing additional software on the managed nodes. For network equipment, this approach is often unacceptable due to limitations in the devices' operating systems. Nornir offers high flexibility through Python, but does not provide built-in mechanisms for managing infrastructure state and requires implementing a significant portion of its functionality.

Ansible uses an agentless architecture and interacts with devices via standard management protocols, including SSH. This allows the platform to be applied to equipment from different manufacturers without modifying its software. A declarative approach to configuration and support for idempotency ensures repeatable operations and keeps devices in a defined state regardless of how many times the scripts are run. A key advantage of Ansible is the availability of specialized network collections that support automated management of Cisco IOS, Cisco ASA, MikroTik, and other manufacturers' equipment. In addition to deploying configurations, the platform allows for collecting the current status of devices, verifying configuration compliance with defined requirements, and automatically resolving detected drifts.

Consequently, Ansible best meets the requirements for building a centralized network device configuration management system that supports security policy compliance monitoring, which led to its selection as the base technology platform for this study.

3. Architecture of security-aware centralized configuration management

The proposed architecture combines IaC principles, centralized configuration storage, and automated security compliance mechanisms to support secure management of network device configurations. The approach has been designed to maintain a consistent configuration baseline across heterogeneous network environments while reducing the operational effort required for configuration deployment, verification, and maintenance.

The architecture incorporates a centralized repository for configuration templates and security policies, an automation platform responsible for orchestration, and a set of managed network devices. Security requirements derived from recognized standards are represented as policies and stored together with configurations in a version-controlled repository. This approach delivers traceability of configuration changes and supports reproducible deployment procedures.

After configuration deployment, the system automatically collects the current device configurations and evaluates them against set security requirements. Any detected drift from the reference state is

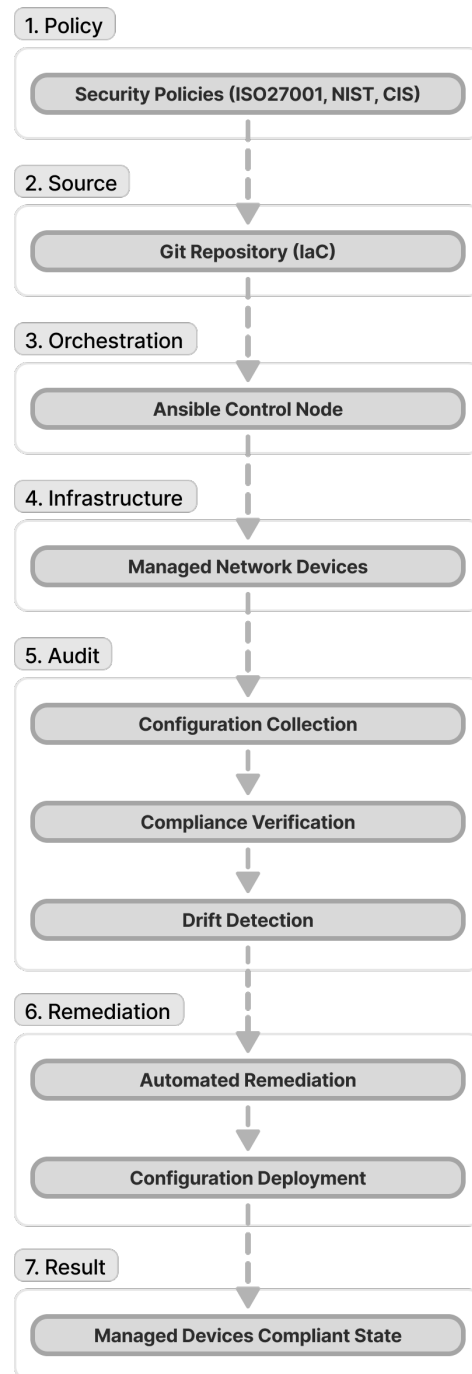


Figure 1: Architecture of Security-Aware Centralized Configuration Management.

treated as configuration drift. In such cases, remediation procedures are automatically initiated to restore compliance with the defined security baseline. The resulting workflow enables continuous configuration validation and automated enforcement of security requirements throughout the configuration management lifecycle. Figure 1 illustrates the architecture pipeline and interaction between the main functional components of the proposed approach.

An important component of the architecture is a formalized set of security policies that define the configuration requirements for network devices. The policies were developed based on the recommendations of ISO/IEC 27001:2022, NIST SP 800-53 Rev. 5, and CIS Critical Security Controls v8 [14, 15, 16]. The implemented set includes policies for access control, control channel protection, event auditing, protection of confidential data, backup, and monitoring of configuration drifts. The mapping of the

Table 1

Mapping of Implemented Security Policies to ISO/IEC 27001:2022, NIST SP 800-53 Rev. 5, and CIS Controls v8

Security Policy	ISO/IEC 27001:2022	NIST SP 800-53 Rev. 5	CIS Critical Security Controls v8
Session Timeout	8.9 Configuration Management	CM-7 Least Functionality	4.3 Configure Automatic Session Locking
Centralized Event Log Collection	8.15 Logging	AU-2 Event Logging	8.2 Collect Audit Logs
Principle of Least Privilege	8.2 Privileged Access Rights	AC-6 Least Privilege	6.1 Establish an Access Granting Process
Automation Secrets Protection	8.24 Use of Cryptography	SC-28 Protection of Information at Rest	4.1 Establish and Maintain a Secure Configuration Process
Management Access Control	8.20 Network Security	AC-17 Remote Access	12.2 Establish and Maintain a Secure Network Architecture
Sensitive Data Encryption	8.24 Use of Cryptography	SC-28 Protection of Information at Rest	3.11 Encrypt Sensitive Data at Rest
Configuration Change Control	8.9 Configuration Management	CM-3 Configuration Change Control	4.1 Establish and Maintain a Secure Configuration Process
Role-Based Access Control	5.15 Access Control	AC-2 Account Management	6.8 Define and Maintain Role-Based Access Control
Centralized Authentication	8.5 Secure Authentication	IA-2 Identification and Authentication (Organizational Users)	6.3 Require MFA for Externally-Exposed Applications
Management Channel Protection	8.20 Network Security	SC-8 Transmission Confidentiality and Integrity	12.3 Securely Manage Network Infrastructure
Configuration Drift Monitoring	8.16 Monitoring Activities	CM-6 Configuration Settings; SI-7 Software, Firmware, and Information Integrity	4.2 Establish and Maintain a Secure Configuration Process for Network Infrastructure; 13.1 Centralize Security Event Alerting
Privileged Access Management	8.2 Privileged Access Rights	AC-6 Least Privilege	6.1 Establish an Access Granting Process
Configuration Backup	8.13 Information Backup	CP-9 Information System Backup	11.2 Perform Automated Backups

implemented policies to the requirements of international standards is shown in Table 1.

To ensure compliance with specified security requirements, the architecture incorporates an automated mechanism for compliance verification and detection of configuration drifts. The monitoring process collects current device configurations, compares them to a baseline state, and verifies adherence to defined security policies. Upon detection of non-compliance, the system generates a violation report and may initiate procedures to restore the configuration to the baseline state. This method reduces the duration that insecure configurations remain active, enhances adherence to security policies, and minimizes the influence of human error in network infrastructure management.

4. Implementation and experimental evaluation

4.1. Experimental environment

To verify the effectiveness of the proposed approach to centralized configuration management for network devices, an experimental environment was deployed using the EVE-NG network emulator. The

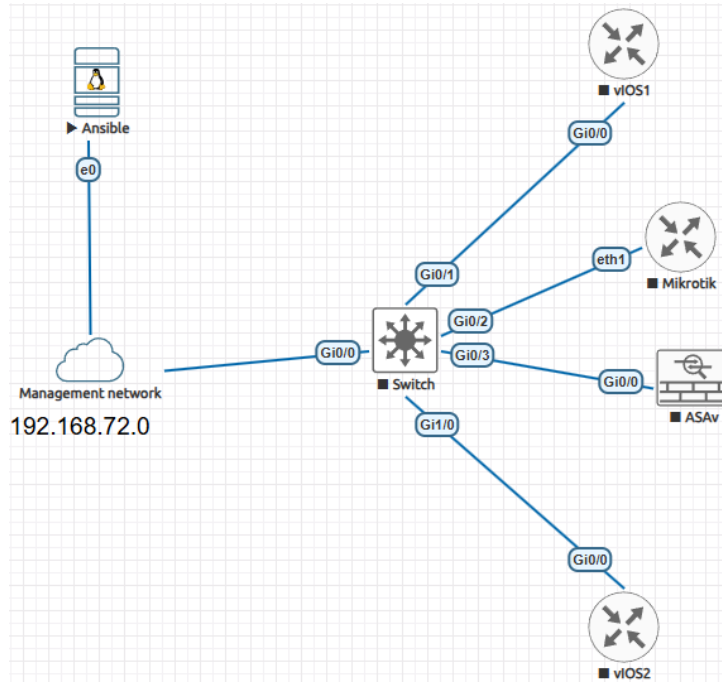


Figure 2: Experimental network topology used for validation of the proposed framework.

Table 2
Experimental Environment Configuration

Device	Role	Management IP	Platform/Image
Ubuntu Server	Ansible Control Node	192.168.72.133	Ubuntu 22.04 LTS
Cisco R1	Router	192.168.72.11	Cisco IOSv
Cisco R2	Router	192.168.72.12	Cisco IOSv
Cisco Switch	Layer 2 Switch	192.168.72.13	Cisco IOSvL2
Cisco ASA	Firewall	192.168.72.14	Cisco ASAv
MikroTik Router	Edge Router	192.168.72.15	MikroTik CHR
EVE-NG Host	Log/NTP Server	192.168.72.10	EVE-NG CE

use of a virtual environment ensures experimental reproducibility and enables simulation of network infrastructure operations without physical hardware.

The experimental setup included an Ansible Control Node running Ubuntu Linux and network devices from various manufacturers. The infrastructure consisted of two Cisco IOSv routers, a Cisco IOSvL2 switch, a Cisco ASAv firewall, and a MikroTik CHR router. This set of equipment allows evaluation of the operation of automation mechanisms in a heterogeneous environment and verification of the applicability of unified security policies to devices across different platforms.

All devices were connected to a separate management network, through which remote access, configuration collection, and the execution of automation scripts were carried out. The logical topology of the experimental environment is shown in Fig. 2.

The main components of the experimental environment are listed in Table 2.

4.2. Implementing security policies using ansible

To implement the proposed architecture, a set of Ansible roles was developed to automate security policy enforcement, verify configuration compliance, and remediate detected non-compliance. All system components are organized using a modular approach, which simplifies maintenance, code reuse, and future functionality extensions. The main components of the implemented system and their

Table 3
Main Components of the Ansible-Based Implementation

Component	Type	Purpose
<code>ansible.cfg</code>	Configuration File	Global automation framework settings
<code>hosts.yaml</code>	Inventory	Managed devices inventory and connection parameters
<code>inventory/group_vars/all/vault.yaml</code>	Encrypted Variables	Protected credentials and sensitive parameters
<code>roles/iam/</code>	Role	Authentication and access control policies
<code>roles/hardening/</code>	Role	Device hardening and secure configuration enforcement
<code>roles/compliance_check/</code>	Role	Automated compliance verification against security policies
<code>roles/remediation/</code>	Role	Automatic correction of detected configuration drifts
<code>roles/backup/</code>	Role	Configuration backup and recovery
<code>results/backups/</code>	Directory	Local storage for configuration backups

purposes are listed in Table 3.

The *Identity and Access Management (IAM) role* is used to implement access control mechanisms. It provides centralized account creation, authentication, and authorization configuration, as well as enforcement of the concept of least privilege. Within this role, the AAA (Authentication, Authorization, and Accounting) mechanism is configured automatically, administrative accounts are created, and the set of available commands is restricted based on the user's role.

The *Hardening role* implements protection for the management plane. It includes policies to minimize the attack surface of network devices. Specifically, the disabling of insecure remote access protocols, the configuration of SSH parameters, the setting of administrative session timeouts, and the restriction of access to management services via access control lists are automated.

Centralized event auditing is provided by a logging mechanism, within which network devices are automatically configured to send event logs to a single log collection server. This approach enables centralized monitoring of configuration changes, detection of suspicious activities, and follow-up analysis of security incidents.

To ensure the ability to restore infrastructure operability after failures or unauthorized changes, the *Backup role* has been implemented. Its primary task is the automatic creation of backups of current network device configurations and their centralized storage. Backups are generated automatically before changes are made and can be used to restore the infrastructure to its previous state.

Special attention has been paid to protecting confidential data. The *Ansible Vault mechanism* is used to store credentials and other sensitive parameters, which ensures the encryption of confidential information using the AES-256 algorithm. As a result, even with access to the configuration repository, an unauthorized user cannot obtain device credentials without knowing the Vault password.

4.3. Configuration drift detection and automated remediation

One of the key challenges in operating network infrastructure is configuration drift, in which devices no longer comply with established security policies or baseline configurations. Such drifts can result from manual changes, administrative errors, or unauthorized interference, leading to violations of information security requirements.

In the proposed architecture, the mechanism for detecting and resolving configuration drift is implemented in accordance with the *Compliance Verification, Drift Detection, and Automated Remediation*

stages shown in Figure 1. Verification is performed by comparing the current state of devices against a set of rules derived from defined security policies.

Automated verification functions are implemented within the `roles/compliance_check/` role. The role begins by collecting the current configuration of network devices using corresponding Ansible modules. For Cisco IOS and Cisco ASA equipment, the `cisco.ios.ios_command` module is used, while MikroTik devices are accessed through the `community.routeros.command` module. The obtained configurations are analyzed by a set of compliance checks, each implementing a separate security policy requirement.

An example of a verification rule is shown in Listing 1. In this case, the rule checks whether SSH version 2 is enabled on a network device. If the verification is successful, the `assert` module generates a message indicating compliance with the requirement. Otherwise, the configuration drift logging mechanism is triggered.

Listing 1: Example of a compliance verification rule

```
- name: "IOS SSH v2"
  block:
    - assert:
        that:
          - "'ip ssh version 2' in current_config.stdout[0]"
        success_msg: "PASS: SSH v2"
        fail_msg: "DRIFT: SSH v2 disabled"
  rescue:
    - set_fact:
        drift_detected: true
    - debug:
        msg: "DRIFT: SSH v2 on {{ inventory_hostname }}"
  when:
    - ansible_network_os == 'cisco.ios.ios'
    - current_config is not skipped
```

During script execution, the results of the checks are processed using the `block/rescue` construct. If any condition is not met, the `drift_detected` flag is set, signaling configuration drift and initiating further actions to resolve it. The result of the drift detection mechanism is shown in Figure 3. In this example, the system detected a configuration mismatch with the security policy and generated a corresponding violation notification.

Upon detecting a drift, the `roles/remediation/` role is triggered, which automatically restores the configuration to its baseline state. The remediation process follows Ansible's declarative approach and reuses the same security policies that were previously applied during device configuration. Because the process is idempotent, repeated execution does not introduce additional modifications when the configuration already complies with the specified requirements.

After the restoration procedure is completed, an automatic re-verification of compliance with security policies is performed. This additional validation confirms the effectiveness of the corrective actions and ensures that the infrastructure has returned to the desired state.

The result of the re-verification after applying the automated remediation mechanism is presented in Figure 4. The `FULLY COMPLIANT` status indicates that no security policy violations remain and that the configuration control cycle has been completed successfully.

Thus, the implemented mechanism provides a fully automated configuration control cycle that includes compliance verification, configuration drift detection, automatic remediation, and subsequent validation of the obtained results. This approach minimizes the influence of human factors and helps maintain a stable level of compliance of the network infrastructure with established security requirements.

4.4. Performance evaluation and discussion

To evaluate the effectiveness of the proposed system, a comparison was conducted between the Ansible-based automated approach and traditional manual management of network devices. During the experi-

```

TASK [compliance_check : assert] *****
skipping: [R1Cisco]
skipping: [R2Cisco]
skipping: [switch]
skipping: [FW]
fatal: [MikT1]: FAILED! => changed=false
  assertion: mt_services.stdout[0] | regex_search('X.*ftp') is not none
  evaluated_to: false
  msg: 'DRIFT: FTP enabled'

TASK [compliance_check : Compliance result] *****
ok: [R1Cisco] =>
  msg: 'R1Cisco: DRIFT DETECTED'
ok: [R2Cisco] =>
  msg: 'R2Cisco: DRIFT DETECTED'
ok: [switch] =>
  msg: 'switch: DRIFT DETECTED'
ok: [FW] =>
  msg: 'FW: DRIFT DETECTED'
ok: [MikT1] =>
  msg: 'MikT1: DRIFT DETECTED'

TASK [test time] *****
ok: [R1Cisco] =>
  msg: 'R1Cisco: 20 sec.'
ok: [switch] =>
  msg: 'switch: 20 sec.'
ok: [R2Cisco] =>
  msg: 'R2Cisco: 20 sec.'
ok: [MikT1] =>
  msg: 'MikT1: 19 sec.'
ok: [FW] =>
  msg: 'FW: 20 sec.'

```

Figure 3: Detection of configuration drift.

```

TASK [Show execution time] *****
ok: [R1Cisco] =>
  msg: Time to fix was57.846 sec.
ok: [MikT1] =>
  msg: Time to fix was57.86 sec.
ok: [switch] =>
  msg: Time to fix was57.831 sec.
ok: [R2Cisco] =>
  msg: Time to fix was57.853 sec.
ok: [FW] =>
  msg: Time to fix was57.909 sec.

TASK [compliance_check : Compliance result] *****
*
ok: [R1Cisco] =>
  msg: 'R1Cisco: FULLY COMPLIANT'
ok: [R2Cisco] =>
  msg: 'R2Cisco: FULLY COMPLIANT'
ok: [MikT1] =>
  msg: 'MikT1: FULLY COMPLIANT'
ok: [switch] =>
  msg: 'switch: FULLY COMPLIANT'
ok: [FW] =>
  msg: 'FW: FULLY COMPLIANT'

PLAY RECAP *****
*
FW                : ok=16   changed=3   unreachable=0   failed=0   skipped=16   rescued=0   ignored=0
MikT1             : ok=17   changed=6   unreachable=0   failed=0   skipped=15   rescued=0   ignored=0
R1Cisco           : ok=26   changed=4   unreachable=0   failed=0   skipped=9    rescued=0   ignored=0
R2Cisco           : ok=26   changed=4   unreachable=0   failed=0   skipped=9    rescued=0   ignored=0
switch            : ok=26   changed=4   unreachable=0   failed=0   skipped=9    rescued=0   ignored=0

```

Figure 4: Successful automated remediation and compliance restoration.

ments, the following were evaluated: configuration deployment time, success rate of applying settings, compliance with security policies, time to detect configuration drifts, time to restore the correct state, and the total duration of the vulnerable configuration.

The summarized results of the experimental study are presented in Table 4, and their graphical comparison is shown in Figure 5. For time-based metrics, a downward change (↓) indicates improvement, whereas for percentage-based metrics, an upward change (↑) indicates improvement.

The findings show a notable advantage of the automated approach over manual administration. The greatest reduction is observed in the configuration deployment time, which decreased more than tenfold due to the parallel execution of operations on multiple devices.

Table 4
Comparison of Average Performance Metrics

Metric	Manual Configuration	Ansible-Based Approach	Relative Change
Configuration deployment time (s)	865	85	↓ 90.1%
Configuration success rate (%)	94	100	↑ 6.4%
Security policy compliance (%)	80	100	↑ 25.0%
Drift detection time (s)	131	46	↓ 65.2%
Remediation time (s)	551	68	↓ 87.6%
Vulnerability exposure time (s)	683	114	↓ 83.3%

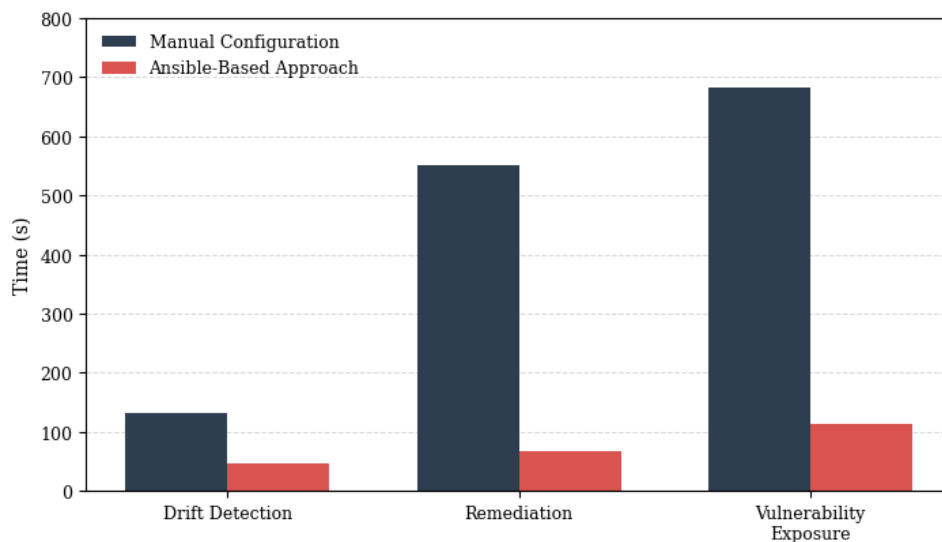


Figure 5: Comparison of Average Performance Metrics

The automated system ensured 100% success in applying configurations and 100% compliance with security policies in all experimental runs. In contrast, manual configuration led to errors and instances of non-compliance with certain security requirements. The time to detect configuration deviations was reduced by nearly three times, and the time to restore the correct state by approximately eight times. As a result, the total lifetime of a vulnerable configuration was reduced by approximately six times, which directly lowers the risk of operating with incorrect settings.

Thus, the proposed approach enables faster configuration management, improves compliance with security policies, and minimizes human error during network infrastructure administration.

5. Conclusions

This paper describes the development and implementation of a centralized network device configuration management system based on Ansible. The proposed approach includes the principles of Infrastructure as Code, automated enforcement of security policies, compliance monitoring, and mechanisms to automatically resolve configuration drift. In addition, a set of security policies for the network infrastructure has been developed and coordinated with the requirements of the international standards ISO/IEC 27001:2022, NIST SP 800-53 Rev. 5, and CIS Controls v8, which ensures the validity of the selected protection mechanisms and simplifies the assessment of the infrastructure's compliance with security requirements.

To verify the proposed approach, an experimental environment was deployed and a series of tests was conducted, covering configuration deployment, verification of compliance with security policies, detection of configuration drifts, and automated restoration of the reference state. The results of the

experimental study confirmed the functionality of the implemented architecture and the effectiveness of automated configuration management versus a manual approach. The results demonstrate the practical suitability of the proposed solution for corporate network infrastructures, where centralized configuration management, monitoring compliance with security policies, and rapid resolution of configuration drift are necessary. A promising direction for further research is the embedding of large language models (LLMs) [17, 18] into the configuration management process to create automation scenarios based on natural-language requirements and to evaluate the causes of detected drifts.

Acknowledgments

This paper was published due to work on the Erasmus+ Project Jean Monnet module “Integrating the future-proof EU cybersecurity ecosystem in Ukraine” Project No.: 101177024 – ERASMUS-JMO-2024-HEI-TCH-RSCH. Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Education and Culture Executive Agency (EACEA). Neither the European Union nor EACEA can be held responsible for them.

Declaration on Generative AI

The authors have not employed any Generative AI tools.

References

- [1] S. T. Arzo, C. Naiga, F. Granelli, R. Bassoli, M. Devetsikiotis, F. H. P. Fitzek, A theoretical discussion and survey of network automation for iot: Challenges and opportunity, *IEEE Internet of Things Journal* 8 (2021) 12021–12045. doi:10.1109/JIOT.2021.3075901.
- [2] P. M. Babaei, M. Kebria, S. Nahavandi, A survey on automation technologies used in network control and management, in: 2022 15th International Conference on Human System Interaction (HSI), Melbourne, Australia, 2022, pp. 1–6. doi:10.1109/HSI55341.2022.9869444.
- [3] F. Pizzato, D. Brighenti, R. Sisto, F. Valenza, Security automation in next-generation networks and cloud environments, in: 2024 IEEE Network Operations and Management Symposium (NOMS), Seoul, Republic of Korea, 2024, pp. 1–4. doi:10.1109/NOMS59830.2024.10575650.
- [4] D. Brighenti, R. Sisto, F. Valenza, Towards security automation in virtual networks, in: 2023 IEEE 9th International Conference on Network Softwarization (NetSoft), Madrid, Spain, 2023, pp. 326–331. doi:10.1109/NetSoft57336.2023.10175459.
- [5] D. Brighenti, G. Marchetto, R. Sisto, F. Valenza, Automation for network security configuration: State of the art and research trends, *ACM Computing Surveys* 56 (2024) 1–37. doi:10.1145/3616401.
- [6] M. Chiari, M. D. Pascalis, M. Pradella, Static analysis of infrastructure as code: A survey, in: 2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C), Honolulu, HI, USA, 2022, pp. 218–225. doi:10.1109/ICSA-C54293.2022.00049.
- [7] A. H. Ibrahim, A. Yousef, W. Medhat, Devsecops: A security model for infrastructure as code over the cloud, in: 2022 2nd International Mobile, Intelligent, and Ubiquitous Computing Conference (MIUCC), Cairo, Egypt, 2022, pp. 284–288. doi:10.1109/MIUCC55081.2022.9781709.
- [8] M. Vihuro, A. Malyar, G. Litawa, K. Kluczevska-Chmielarz, T. Konrad, P. Migo, Lightweight semantic segmentation for fermentation foam monitoring: A comparative study of u-net, deeplabv3+, fast-scnn, and segnet, *Applied Sciences* 16 (2026) 1487. doi:10.3390/app16031487.
- [9] E. Billoir, R. Laborde, A. S. Wazan, Y. Rutschle, A. Benzekri, Enhancing secure deployment with ansible: A focus on least privilege and automation for linux, in: Proceedings of the 19th International Conference on Availability, Reliability and Security (ARES '24), Association for Computing Machinery, New York, NY, USA, 2024, pp. 1–7. doi:10.1145/3664476.3670929.

- [10] R. Ulandari, B. Arifwidodo, J. G. A. Ginting, Ansible optimization for network configuration: A performance comparison on routing protocols, in: 2025 12th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI), Semarang, Indonesia, 2025, pp. 798–803. doi:10.1109/EECSI67060.2025.11290162.
- [11] Y. Fan, X. Lu, W. Chen, Design and implementation of a python-based network security automation platform, in: 2025 International Conference on Computer, Internet of Things and Smart City (CIoTSC), Suzhou, China, 2025, pp. 1–7. doi:10.1109/CIoTSC67482.2025.11413115.
- [12] S. V. Georgiev, K. S. Nikolova, An innovative comparison of netdevops configuration management solutions for automation of data center networks, in: 2025 60th International Scientific Conference on Information, Communication and Energy Systems and Technologies (ICEST), Ohrid, North Macedonia, 2025, pp. 1–4. doi:10.1109/ICEST66328.2025.11098348.
- [13] Complete guide to nornir network automation for engineers, 2025. URL: <https://netodata.io/complete-guide-to-nornir-network-automation-for-engineers/>.
- [14] Iso/iec 27001:2022 information security, cybersecurity and privacy protection – information security management systems – requirements, 2022. URL: <https://www.iso.org/standard/27001>.
- [15] Nist sp 800-53 rev. 5: Security and privacy controls for information systems and organizations, 2020. URL: <https://doi.org/10.6028/NIST.SP.800-53r5>. doi:10.6028/NIST.SP.800-53r5.
- [16] Cis critical security controls version 8, 2021. URL: <https://www.cisecurity.org/controls/v8>.
- [17] D. Boinpally, Generative ai-powered infrastructure-as-code: Automating cloud setup with explainability and governance, in: 2026 IEEE International Conference on Emerging Computing and Intelligent Technologies (ICoECIT), Hyderabad, India, 2026, pp. 1–7. doi:10.1109/ICoECIT68303.2026.11497432.
- [18] R. Hase, Y. Wang, T. Koike-Akino, J. Liu, K. Parsons, J. Hato, Evaluating security policy compliance in infrastructure as code generated by large language models, in: 2026 14th International Symposium on Digital Forensics and Security (ISDFS), Boston, MA, USA, 2026, pp. 1–6. doi:10.1109/ISDFS69419.2026.11458930.