Experimenting With Security Compliance Checking Using ReFrame

Victor Holanda Rusu
victor.holanda@cscs.ch
ETH Zurich, Swiss National Supercomputing Centre (CSCS)
Lugano, Switzerland

Matteo Basso
matteo.basso@cscs.ch
ETH Zurich, Swiss National Supercomputing Centre (CSCS)
Lugano, Switzerland

Chris Gamboni
chris.gamboni@cscs.ch
ETH Zurich, Swiss National Supercomputing Centre (CSCS)
Lugano, Switzerland

Fabio Zambrino
fabio.zambrino@cscs.ch
ETH Zurich, Swiss National Supercomputing Centre (CSCS)
Lugano, Switzerland

Massimo Benini
benini@cscs.ch
ETH Zurich, Swiss National Supercomputing Centre (CSCS)
Lugano, Switzerland

ABSTRACT

Security compliance is a growing concern in High-Performance Computing (HPC) environments, where the complexity and scale of infrastructure pose unique challenges to maintaining system integrity. Traditional compliance tools are often seen as intrusive or disconnected from the workflows of HPC engineers. In this work, we explore the use of ReFrame, a regression testing framework widely adopted in HPC, for implementing and managing security compliance checks. Leveraging the community-maintained ComplianceAsCode project, we generate portable, programmable compliance tests that integrate seamlessly into existing engineering practices. Our approach enables a shift-left security model, a practice that advocates moving security considerations earlier in the development and deployment pipeline, by embedding compliance into routine system validation, promoting early detection of misconfigurations that may impact not only security but also system performance and reliability. We evaluate our implementation on RHEL 9.5 and openSUSE Leap 15.4, demonstrating minimal performance overhead and strong alignment with OSCAP results. This work illustrates how familiar tools can be extended to meet evolving security needs without compromising usability, and sets the foundation for future community-driven enhancements in compliance automation.

CCS CONCEPTS

• Security and privacy → Systems security; **Operating systems security**; **Intrusion/anomaly detection and malware mitigation**; **Security requirements**; **Malware and its mitigation**; **Intrusion detection systems**; **• Information systems → Storage management**; **Computing platforms**; **Enterprise information systems**; **Open source software**;

KEYWORDS

HPC Security, System Security, Security Compliance, Compliance Checking, ReFrame, Regression Testing

ACM Reference Format:
Victor Holanda Rusu, Matteo Basso, Chris Gamboni, Fabio Zambrino, and Massimo Benini. 2018. Experimenting With Security Compliance Checking Using ReFrame. In *Proceedings of Computing Horizons: CUG 2025 (CUG25).* ACM, New York, NY, USA, 8 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

1.1 Background and Motivation

High-Performance Computing (HPC) systems are at the heart of scientific breakthroughs, defense simulations, climate modeling, biological applications, and increasingly, commercial innovation. Given the strategic importance of the data and processes running on these systems, security is no longer an optional consideration, it is fundamental. The unique scale and complexity of HPC environments introduce specific challenges, such as shared multi-user access, high-throughput job scheduling, and the integration of diverse software stacks. Any security lapse can lead to the leakage of sensitive research data, theft of intellectual property, or even disruptions to national infrastructure.

As HPC systems become more interconnected with cloud resources and external collaborators, the attack surface grows, making robust security practices essential for operational integrity and trust. To manage this complexity, organizations increasingly rely on best practices in security that are formalized through internal policies, external standards, and industry-recognized baselines [4]. This helps ensure consistency in large and diverse environments and acts as guiding principles for what “secure” should look like in practice. In this context, compliance testing plays a role in validating whether specific predefined criteria, derived from these standards and policies, are being met. However, it should not be conflated with the broader cybersecurity discipline.

1.1.1 Standard security compliance checks are not enough. Compliance checks are essentially audits; they verify that specific controls are present and configured according to a standard. However, this snapshot view does not account for many other aspects such as evolving threats, insider risks, or unknown vulnerabilities.

Security must be seen as a dynamic and adaptive process: compliance is just one layer. Real-world attacks often succeed not because organizations failed compliance but because they lacked active monitoring, incident response, or threat intelligence integration. Thus, security and compliance are complementary, not synonymous.

Security compliance frameworks are grounded in formal regulatory or certification requirements, which are designed to ensure that systems meet minimum security expectations for particular industries or locations. These frameworks translate legal or ethical obligations into technical and administrative controls, such as access restrictions, data encryption, and audit logs. At its core, compliance measures how well an organization's security practices align with a defined standard. These standards act as benchmarks, helping to assess the presence, configuration, and effectiveness of specific controls. For example, compliance with ISO/IEC 27001 [13] involves demonstrating a formalized risk management process and verifying that critical assets are protected according to the organization's security policy. It provides structure and documentation, making security efforts more transparent and auditable. However, measurement does not equate to effectiveness, it's possible to meet the letter of a standard while failing to achieve meaningful security outcomes.

1.1.2 *Security standards provide a good basis for the implemention of cybersecurity controls.*

The benefit of the security standards used in compliance frameworks is that they originate from regulatory bodies, industry groups, or international organizations that define best practices for information security. Some of these standards encapsulate decades of collective expertise and are periodically updated to reflect changes in technology, threats, and risk management strategies. They may be prescriptive, detailing exact controls to implement, or risk-based, allowing organizations to tailor their approach.

Either way, the source of the standard, be it NIST [16], ISO [12], or sector-specific frameworks—provides legitimacy and a shared foundation for measuring security maturity.

Compliance frameworks are designed to ensure a baseline level of protection. This is important for establishing consistency across organizations and industries. However, these frameworks are not ceilings, they are floors.

Organizations are encouraged to build upon them by incorporating additional controls that reflect their unique operational realities, risk tolerance, and business priorities.

For example, a research institution handling sensitive genomics data may choose to implement stronger data segregation policies than those required by basic frameworks. We see compliance as a starting point, not an endpoint. This is key to achieving true security resilience.

1.1.3 *Security compliance does not prevent attacks but helps avoid penalties and legal exposure.*

Compliance should be viewed as a form of minimal due diligence, a way to prove that reasonable efforts were made to secure systems. While compliance itself does not prevent breaches, it can significantly influence the consequences of a breach. Demonstrating compliance can reduce liability, soften reputational damage, and serve as a legal defense, in the event of a security breach. Conversely, non-compliance can lead to regulatory fines, lawsuits, and loss of public trust, even in the absence of a security incident. Thus, compliance acts as both a protective and preventive legal measure, not a direct security defense mechanism.

1.1.4 *Security compliance must exist, even for non-regulated organizations.*

Even if an organization is not subject to specific regulations, it must define its own internal compliance standards. Without a shared baseline, teams cannot coordinate effectively, and it becomes impossible to verify whether fundamental protections are in place. Internal standards help enforce consistency across teams and ensure that all systems, whether customer-facing or internal, meet an agreed-upon level of security. This is especially critical in research institutions or forefront HPC environments, where rapid development cycles can lead to security being deprioritized in the absence of formal guidance.

Without a shared, minimum baseline of security controls, each team may implement their own solutions in an inconsistent or even conflicting manner. This siloed approach leads to fragmentation, blind spots, and gaps in coverage. Worse, it prevents organizations from having a unified incident response or from accurately assessing their overall security posture. A baseline ensures that everyone, from system engineers to DevOps teams, works toward the same security objectives, using interoperable and complementary controls. It is the foundation upon which mature security practices are built.

An organization without any formal compliance standard is effectively operating without a security compass. This not only increases the risk of breach but also creates significant exposure in the event of an incident. Regulators, partners, insurers, and even customers expect organizations to meet at least a minimal level of due diligence.

A lack of compliance suggests negligence or immaturity, and it can result in severe reputational and financial consequences, even if no laws were technically broken. As a result, compliance isn't just for regulated sectors, it's a universal marker of operational responsibility.

1.1.5 *Maintaining effective compliance in HPC environments is no small feat.*

Developing security compliance tests is a complex and resource-intensive task, which requires significant engineering knowledge and effort to design, implement, and maintain effective solutions.

Engineers must write, maintain, and automate tests that can validate a wide range of security behaviors, access controls, logging, data handling, user privileges, and more, across complex, distributed systems. These tests must also be resilient to software and hardware changes, and flexible enough to adapt to evolving threats. This requires deep collaboration between security, infrastructure, and application teams, and a shared commitment to building testing into the development and deployment lifecycle.

While tools like OpenSCAP (OSCAP) [18] offer vendor and Linux distribution support for different industry standards, their applicability varies across distributions, with some being well-supported while others lack comprehensive coverage. This inconsistency poses challenges for achieving uniform security compliance across a diverse HPC infrastructure. The predefined set of tests provided by such tools often needs to be expanded or customized to meet the specific requirements of individual computing centers. This customization demands programmability and flexibility within the testing framework to ensure ease of adoption and compatibility across different environments.

Additionally, Open Source and freely available security-focused tools, such as OSCAP, are less known and accessible to engineers from other disciplines, like performance optimization or system operations, who may not be familiar with their functionality or utility. This creates a barrier to collaborative efforts in improving system-wide configurations and promoting shift-left security in HPC centers, a practice that advocates moving security considerations earlier in the development and deployment pipeline. These tools tend to offer limited programmability and flexibility that further constrain its applicability, as users cannot easily customize or extend its capabilities to address unique or evolving HPC use cases.

1.1.6 Security compliance is like regression testing, it detects when things break. Just as software teams use regression testing to ensure that new code changes haven't broken existing functionality, security teams must continually test their controls to ensure that updates, deployments, or configuration changes haven't degraded security. This is particularly important in HPC, where performance tuning and architectural changes are common. Without regular security compliance checks, it's easy for critical controls to be silently disabled or bypassed. These regressions may not show up until a breach occurs, by which time it's too late.

These challenges are strikingly similar to those addressed by performance regression frameworks like ReFrame. [15] Security compliance tests, much like the regression and performance tests, must be adaptable to the unique configurations and requirements of individual systems and centers.

The complexity of managing heterogeneous hardware, varying software stacks, different Linux distributions, and diverse operational policies in HPC environments mirrors the issues tackled by ReFrame's system-independent design.

ReFrame does not include a predefined set of security compliance tests, which presents an opportunity to develop and integrate such tests into its framework. It is possible to create comprehensive and tailored security compliance checks that align with established standards to evaluate system configuration, generic exploits, assess different application contexts, and execute tests in parallel, optimizing testing workflows without significant performance penalties.

1.1.7 Many tools can be used to perform security compliance checks. While ReFrame is a powerful and widely adopted tool for infrastructure and system validation, especially within HPC environments, it is far from being the only option available. The ecosystem of infrastructure testing tools is diverse, spanning open-source projects, commercially supported solutions, and hybrid approaches that offer different trade-offs in flexibility, scalability, and integration. Here, we present a non-exhaustive list of tools that can be used for security compliance.

Among open-source solutions, Pavilion 2 [19] stands out for its focus on scalable performance testing within HPC clusters. It is designed to handle large test suites and integrates tightly with workload managers, making it suitable for continuous performance validation in complex environments.

Tools like Testinfra [26] and Serverspec [25] are infrastructure testing frameworks that integrate well with configuration management systems like Ansible [1], Puppet [21], and Chef [5]. Testinfra uses Python with a pytest-like interface [22], making it easy to write tests for verifying system state after provisioning. Serverspec, on the other hand, uses Ruby [24] and offers an RSpec-style [23] syntax to check whether a system is configured correctly. Both tools are excellent for enforcing infrastructure-as-code practices and ensuring system consistency.

Goss [11] is another lightweight and fast tool, particularly suited for DevOps pipelines and containerized environments. Goss uses YAML [28] files for test definitions and supports a simple, declarative syntax. Its speed makes it a favorite for embedding into Docker builds or CI/CD pipelines to verify that base images meet security and configuration expectations.

For more security-focused infrastructure assessment, there are tools like THOR Lite [17], which evolved from the earlier open-source project Loki [9]. These tools specialize in threat detection, scanning systems for indicators of compromise (IOCs), suspicious files, and signs of known malware. THOR Lite is often used in security operations for proactive threat hunting and forensic analysis.

In the realm of compliance testing, Lynis [6] stands out as a long-established tool. It performs in-depth system and security audits and provides recommendations for hardening. Lynis supports compliance with standards like ISO/IEC 27001 [13], PCI-DSS [20], and HIPAA [8], making it a useful choice for regulated environments that need to demonstrate adherence to external security requirements.

Similarly, OSCAP is a powerful suite for security compliance auditing. It implements the Security Content Automation Protocol (SCAP) [14] and supports automation of configuration and vulnerability scanning using recognized benchmarks such as those published by NIST [16] and CIS [3]. OSCAP is widely used in both government and enterprise environments to ensure that systems meet formal security and compliance policies.

Each of these tools comes with its own advantages and limitations, some excel in performance and cluster-oriented testing, others in compliance, and some in infrastructure as code validation. The diversity of tools reflects the diversity of infrastructure challenges, and as is often the case in software tooling, there are no silver bullets, the right tool is context-dependent, shaped by system architecture, user expertise, organizational goals, environment complexity, and existing workflows.

When choosing a tool for security compliance checks, it is essential to prioritize not just the technical capabilities of the tool, but also its accessibility and usability for the engineers who will be responsible for running, maintaining, and evolving it. In complex, multidisciplinary teams such as those found in HPC sites, the learning curve and adoption friction of a new tool can significantly affect its long-term success. Therefore, selecting a tool that is already familiar to most engineers, or one that is easy to learn, extend, and integrate, is imperative for ensuring broad participation and consistent application of security policies.

This ease of use and alignment with team workflows becomes particularly important in the context of shift-left security. In the HPC domain, where system complexity and performance sensitivity are high, shift-left security means that every engineer, not just dedicated security staff, takes responsibility for applying security best practices and executing security tests. This distributed model of accountability requires tools that lower the barrier to entry for writing, running, and debugging tests, thereby encouraging engineers to incorporate security into their everyday workflows.

Our choice of ReFrame was driven by its existing adoption at CSCS, its Python-based programmability, and the familiarity it offers to engineering teams at CSCS. It was not driven by the fact that its development started at CSCS. The familiarity with the tool helps address the cultural challenge of compliance being perceived as a “check-the-box” exercise, instead integrating it as a natural part of system validation, which is our main focus.

1.2 Our Contributions

This paper proposes an alternative approach by adapting the ReFrame framework, originally designed for HPC performance and regression testing, to perform security compliance testing. We show tests derived from the ComplianceAsCode project [7], structured and executed within ReFrame, and deployed in privileged and non-privileged contexts for on three different security standards STIG DoD [27], ANSSI BP-28-enhanced [2], and CSCS’ internal one.

Our contributions include:

- A method for converting ComplianceAsCode rules into ReFrame tests.
- Integration strategies that maintain performance and usability.
- A discussion of implementation choices and their impact on maintainability.
- Results from running the system on RHEL 9.5 and openSUSE Leap 15.6.

2 LANDSCAPE OF SECURITY COMPLIANCE TOOLS

Numerous tools exist for infrastructure and compliance validation, ranging from open-source to commercial offerings. In this paper, we focus our attention to a limited set of open source tools that we have used or have seeing in use in other HPC centers. Pavilion 2 [19], Testinfra [26], Serverspec [25], Goss [11], OSCAP, and Lynis [6] are examples of offer different models for defining and executing tests. These tools vary in portability, ease of use, and level of abstraction.

Among open-source solutions, Pavilion 2 [19] stands out for its focus on scalable performance testing within HPC clusters. It is designed to handle large test suites and integrates tightly with workload managers, making it suitable for continuous performance validation in complex environments.

Testinfra and Serverspec are infrastructure testing frameworks that integrate well with configuration management systems like Ansible [1], Puppet [21], and Chef [5]. Testinfra uses Python with a pytest-like interface [22], making it easy to write tests for verifying system state after provisioning. Serverspec, on the other hand, uses Ruby [24] and offers an RSpec-style [23] syntax to check whether a system is configured correctly. Both tools are excellent for enforcing infrastructure-as-code practices and ensuring system consistency.

Goss is another lightweight and fast tool, particularly suited for DevOps pipelines and containerized environments. Goss uses YAML [28] files for test definitions and supports a simple, declarative syntax. Its speed makes it a favorite for embedding into Docker builds or CI/CD pipelines to verify that base images meet security and configuration expectations.

In the realm of compliance testing, Lynis [6] is as a long-established tool. It performs in-depth system and security audits and provides recommendations for hardening. Lynis supports compliance with standards like ISO/IEC 27001 [13], PCI-DSS [20], and HIPAA [8], making it a useful choice for regulated environments that need to demonstrate adherence to external security requirements.

Our choice of ReFrame was driven by its existing adoption at CSCS, its Python-based programmability, and the familiarity it offers to engineering teams, it was not driven by the fact that its development started at CSCS. The familiarity with the tool helps address the cultural challenge of compliance being perceived as a “check-the-box” exercise, instead integrating it as a natural part of system validation. While we continue to evaluate and occasionally incorporate complementary tools, ReFrame remains at the core of our infrastructure testing strategy due to its alignment with our operational needs and HPC focus.

3 COMPLIANCE CHECKS DESIGN PRINCIPLES

3.1 Shift-Left Security in HPC

To enable effective security, compliance testing should be embedded early in the system lifecycle. This shift-left approach means engineers themselves are responsible for defining and maintaining compliance logic. For this to succeed, the framework must be approachable, extensible, and integrate well with existing tools.

3.2 Read-Only and Modular Tests

All tests are designed to be non-intrusive. They operate in a read-only mode, ensuring no accidental changes to the system. Each test inherits from a base class that enforces common behaviors, including the ability to toggle verbosity or failure diagnostics via command line interface (CLI).

Each compliance requirement is implemented as a dedicated test. While this limits reuse via parameterization, it allows fine-grained tagging, traceability, and control over individual rules.

3.3 Privilege Separation

Tests requiring elevated privileges inherit from a mixin that checks for root access. If not running in a privileged context, they are skipped. Similarly, tests designed for non-root execution will skip themselves if run as root. Tests that can be executed in both contexts will not inherent the mixin enforcing user permissions. This separation facilitates safe testing, avoids misconfiguration and unnecessary failures.

4 IMPLEMENTATION DETAILS

4.1 ComplianceAsCode Integration

We based our test generation on the ComplianceAsCode project, which provides policy-to-rule mappings, Ansible roles, and SCAP-compatible content. While the project primarily targets RHEL, we extended support to openSUSE by adapting paths and logic as needed.

Tests were generated using the controls and rules YAML files and rendered into ReFrame test templates. A layer of automation was used to convert these rules into Python classes, which were then manually curated to use ReFrame features such as mixins, parameterization, and dependency management.

4.2 Execution Logic and Policy

Tests are tagged according to severity, OS, privilege level, and optionally the execution context. ReFrame's CLI allows selective execution based on these tags. Tests that scan external filesystems may impact system performance and are run separately, usually on dedicated systems.

5 RESULTS AND DISCUSSION

5.1 Setup and Scope

Due to security and confidentiality considerations, we cannot disclose or share the detailed results of our compliance testing experiments performed on production-level CSCS systems. These systems represent critical infrastructure, and as such, any technical details, must remain restricted to protect operational integrity and sensitive data. Nevertheless, the methodologies, tools, and lessons learned from those experiments can be generalized and discussed using analogous test environments. Here we show results for the evaluated of our framework using VMs running RHEL 9.5 and openSUSE Leap 15.6. These testbeds were used during the RHEL 7 to RHEL 9 migration effort, offering an ideal scenario to validate compliance coverage and system behavior.

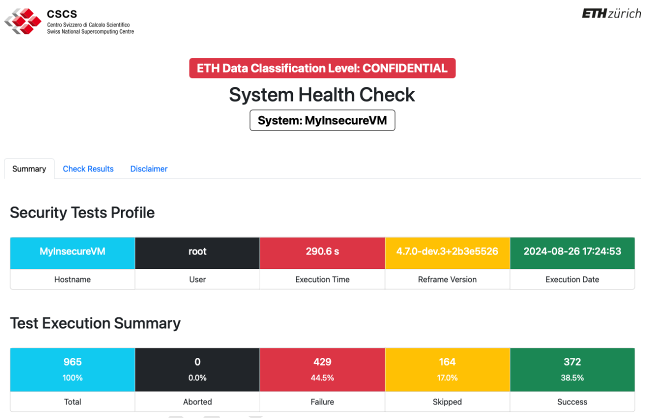
5.2 Observations

We can summarise our findings as follows. First, the programmability of the solution allowed for straightforward adaptation to meet CSCS-specific requirements, demonstrating its flexibility in real-world deployments. Second, the integration process did not introduce any noticeable increase in system boot time, which is a critical factor for production environments. Third, performance-impacting tests, such as parallel filesystems scanning, were effectively isolated and scheduled to run independently, ensuring they did not interfere with normal operations. Finally, adopting a tool familiar to our enginees, makes it easier to integrate security checks directly into our existing regression test infrastructure, as well as practical and maintainable, helping to reduce the natural reluctance often associated with security-related topics.

5.3 RHEL 7 to RHEL 9 migration

One particularly important benefit during the RHEL 7 to RHEL 9 migration was the customization that can be achieved when using a programmable framework known by the engineers. The familiarity with ReFrame helped bridge the gap between compliance and practical hardening. During the migration we could extend our compliance checks, extending our coverage significantly. This very important to us because we needed to validate not just generic compliance benchmarks, but also local policies and configuration nuances unique to our center's operational and security model.

Figure 1 shows the landing page of the Compliance Check report generated for a test VM. In this particular case, we tampered the test VM partially in order to display the output of the report when tests fail.


CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre
ETH zürich
ETH Data Classification Level: CONFIDENTIAL

System Health Check

System: MyInsecureVM

Summary
Check Results
Disclaimer

Security Tests Profile

MyInsecureVM	root	290.6 s	4.7.0-dev.3+2b3e5526	2024-08-26 17:24:53
Hostname	User	Execution Time	Reframe Version	Execution Date

Test Execution Summary

965 100%	0 0.0%	429 44.5%	164 17.0%	372 38.5%
Total	Aborted	Failure	Skipped	Success

Figure 1: From page of the Compliance Check report generated for a test VM.

The current test suite includes approximately 1,200 supported compliance tests, offering a comprehensive coverage of policy controls derived from the ComplianceAsCode project and CSCS-specific requirements. These tests span multiple categories, from filesystem permissions to package integrity, user account policies, and system configurations. This breadth allows system administrators to validate a wide array of security requirements and use cases.

5.4 Performance implications

The performance measurements illustrated in Figure 1 were obtained by configuring ReFrame to run up to four tests concurrently. This parallel execution helps significantly reduce the total wall-clock time for large-scale test suites, especially when individual tests are I/O bound or can run independently.

It is also worth noting that only a limited subset of tests requires interaction with the workload manager, such as SLURM. Most compliance checks run directly on the system and do not require job scheduling, which makes them ideal candidates for execution on individual compute nodes, either as part of boot-time validation, health checks, or during maintenance windows.

When executed during the boot process as the last daemon, there is no perceived increased in the boot time, because the compliance checks are performed after the system is in an accessible state. In this case, we avoid running tests that scan external filesystems because they are executed in specific systems and sequentially.

Among all the available tests, two particular checks stand out in terms of execution time: `rpm_verify_hashes_check` and `rpm_verify_permissions_check`. These tests evaluate the integrity and permission settings of all installed RPM packages, which involves accessing and validating a large number of files across the system. As a result, they take considerably more time to complete (approximately 245 seconds in our testing VM) than other checks, which accounts for the majority of the total execution time of a compliance run.

5.5 Comparison with OSCAP

We compared ReFrame-based tests with OSCAP scans and found comparable coverage. However, ReFrame allowed for clearer diagnostics, better integration with engineering tools, and easier debugging by system engineers.

Figure 2 shows the a XML snippet defines a security compliance rule in SCAP/XCCDF format that ensures the `/etc/passwd` file is group-owned by `root`. It is part of the ComplianceAsCode content and is identified by the rule ID defined in the `<Rule ID>` field. The `<check>` element references an OCIL script that describes how to perform the check, while the `<fix>` element provides a remediation action using an Ansible playbook. This structured format allows tools like OSCAP to automate the validation and remediation of system configurations based on defined security policies.

```
    <Rule id="xccdf_org.ssgproject.
        ↪ content_rule_file_groupowner_etc_passwd">
    <title>Ensure group ownership of /etc/passwd file is
        ↪ root</title>
    <description>
      The /etc/passwd file should be group-owned by root
          ↪ to prevent
      unauthorized users from obtaining sensitive
          ↪ information.
    </description>
    <check system="urn:xccdf:check:system:ocil">
      <check-content-ref href="checks/ocil/
          ↪ file_groupowner_etc_passwd.xml"/>
    </check>
    <fix system="urn:xccdf:fix:system:ansible">
      <fix-content-ref name="
          ↪ fix_file_groupowner_etc_passwd_ansible"/>
    </fix>
  </Rule>

  <!-- Associated OCIL content -->
  <ocil:questionnaire id="
      ↪ file_groupowner_etc_passwd_questionnaire">
    <ocil:question id="
        ↪ file_groupowner_etc_passwd_question">
      Is the group owner of the /etc/passwd file set to '
          ↪ root'?
    </ocil:question>
    <ocil:action id="file_groupowner_etc_passwd_action">
      To verify the group ownership of the /etc/passwd
          ↪ file, run the following command:
      <ocil:command>stat -c "%G" /etc/passwd</
          ↪ ocil:command>
      The output should be:
      <ocil:command_output>root</ocil:command_output>
      If the group owner is not 'root', this is a finding
          ↪ .
    </ocil:action>
  </ocil:questionnaire>
```

Figure 2: XCCDF rule and associated OCIL check for verifying that `/etc/passwd` is group-owned by `root`. The rule defines the compliance requirement, and the OCIL content provides human-readable instructions for manual verification.

Similarly, Figure 3 shows the same test implemented using ReFrame. In this case, the `paths` parameter in the ReFrame tests serves as a flexible and extensible data structure for defining expected file or directory properties. It is implemented as a list of dictionaries, where each dictionary maps a file or directory path to a set of expected attributes, such as ownership, group, or permission mode. This structure allows the test to handle multiple compliance checks in a uniform way, enabling the validation of a variety of system objects within a single test instance. Here, the test asserts that the `/etc/passwd` file must have `root` as its group owner. The schema supports additional attributes, allowing test developers to compose detailed compliance assertions declaratively. The variables `descr`, `rationale`, `severity`, and `correction` are used to document the context and reasoning of the particular test and generate reports encoding this information. The test also define a wide array of compliance tags. These tags link the check to multiple regulatory and security frameworks (e.g., CIS, PCI-DSS, ISO 27001, NIST), highlighting the rule's relevance across standards, as well as allowing the use of tags to select which type of tests to run via ReFrame's CLI.

```
@rfm.simple_test
class file_groupowner_etc_passwd_check(seccommon.
    ↪ file_permissions_and_ownership_check):
    paths = parameter([{'/etc/passwd' : {'group' : 'root'
        ↪ }}])
    descr = 'Ensure group ownership of /etc/passwd file
        ↪ is root'
    rationale = 'The /etc/passwd file contains
        ↪ information about the users that are
        ↪ configured on the system. Protection of this
        ↪ file is critical for system security.'
    severity = 'medium'
    correction = 'automated'
    tags |= {'cce@sle15', '
        ↪ permissions_important_account_files', 'isa
        ↪ -62443-2009', 'system', 'cis@sle15', 'pcidss'
        ↪ , 'ANSSI-BP-028', 'CCE-91627-0', 'RHEL
        ↪ -09-232135', 'CCE-80798-2', 'anssi', 'CCE
        ↪ -26639-5', 'iso27001-2013', 'cis@sle12', 'isa
        ↪ -62443-2013', 'cce@rhel7', 'nist', 'R50', '
        ↪ cjis', 'intermediary', 'cce@rhel9', 'srg', '
        ↪ cis-csc', 'cis@ubuntu2204', 'cobit5', 'nerc-
        ↪ cip', 'cce@rhel8', 'files', 'medium', '
        ↪ permissions', 'cce@sle12', 'cis@ubuntu2004',
        ↪ 'CCE-83950-6', 'CCE-85809-2', 'nist-csf', '
        ↪ stig_rhel9', 'automated'}
```

Figure 3: ReFrame compliance check for validating the group ownership of `/etc/passwd`.

Both the OSCAP and ReFrame implementations shown in Figures 2 and 3 serve the purpose of validating that `/etc/passwd` is group-owned by `root` user. However, they differ significantly in design philosophy, integration flexibility, and target audience. The OSCAP approach is declarative and based on static XML content, where rules (XCCDF) are paired with checks (OCIL) and optional fixes (e.g., Ansible snippets). While this structure is standardized and widely adopted across enterprise and government environments, it lacks programmability and is less adaptable to the dynamic and complex setups commonly found in HPC systems. This approach also requires the use of BASH [10] to extract the information required.

In contrast, the ReFrame-based solution embeds compliance checks directly into a programmable, Pythonic test framework, e.g., the file permissions for the example shown in Figure 3 are acquired using Python's `os.stat` and `stat.filemode` functions (implementation is not shown). Moreover, ReFrame enables these checks to coexist with performance and regression tests, making them part of routine engineering workflows rather than standalone security audits. Metadata such as severity, rationale, and compliance mappings are incorporated directly into the test definitions, improving traceability and maintainability. As a result, the ReFrame approach lowers the barrier for adoption among HPC engineers while enabling tighter integration with CI/CD pipelines and operational tooling.

One of the benefits of integrating these tests with ReFrame is the ability to dynamically select which tests to execute using command-line flags. Through ReFrame's tagging and parameterization system, users can filter tests based on tags like operating system, privilege requirements, severity, or custom-defined groups. Figure 1 shows a set of 965 tests executed on a test VM. This makes it straightforward to tailor each test run to the specific validation needed, whether it's a lightweight check after system updates or a full compliance scan during a security audit. This idea was borrowed from our experience running ReFrame daily in production, post maintenance and in Continuous Integration and Continuous Deployment environments. There we also use tags to select the right set of tests to be performed to meet the required business goals.

Among all the available tests, two particular checks stand out in terms of execution time: `rpm_verify_hashes_check` and `rpm_verify_permissions_check`. These tests evaluate the integrity and permission settings of all installed RPM packages, which involves accessing and validating a large number of files across the system. As a result, they take considerably more time to complete (approximately 245 seconds in our testing VM) than other checks, and in many environments, they may account for the majority of the total execution time of a compliance run.

5.6 Beyond Security

The return on investment for implementing security compliance checks is not always immediately apparent, which can make it difficult for engineering teams and management to recognize their long-term value. These checks are often viewed as a cost center rather than a strategic investment, primarily because they are associated with regulatory pressure or risk mitigation rather than direct productivity gains. However, this perspective overlooks the broader benefits that security compliance checks can offer beyond their immediate scope.

Security compliance checks often require visibility into and validation of a wide array of system components. This broad reach means that they can inadvertently expose configuration issues, inefficiencies, or oversights that affect other areas of system operations, such as performance bottlenecks, system instability, or interoperability concerns. For example, ensuring synchronized clocks helps with logging, debugging, and monitoring, not just security.

5.7 Normalizing Compliance into Engineering

By embedding compliance tests into the same frameworks used for performance and regression validation, they become part of the engineering workflow. This promotes a cultural shift that encourages greater acceptance and engagement from technical teams, who begin to see these checks as part of their toolkit for building and maintaining robust and reliable systems, not an additional burden imposed by security theoreticians and bureaucrats. This reduces resistance and reframes security as part of system quality, rather than a blocker.

Our choice of ReFrame was driven by its existing adoption at CSCS, so additional burden to maintain yet another tool was key for the integration of security checks. Other centers are encouraged to adopt tools that their engineering team feel more confortable improving the adoption and support of security testing into their own environments. It is important to remember that the right tool speaks your team's language, cutting through noise to focus on the security that actually matters.

6 FUTURE WORK

As we look toward the future of our compliance testing framework, there are several areas that present both opportunities and challenges.

- Expand coverage to include all ComplianceAsCode rules.
- Generalize test logic to support other Linux distributions.
- Open-source the tooling that converts ComplianceAsCode controls to ReFrame.
- Open-source the framework by removing CSCS-specific paths.
- Improve support for non-privileged execution and reporting.
- Maybe Link ReFrame as a backend to ComplianceAsCode directly.
- Share tooling for artifact collection and audit reporting.

7 CONCLUSION

Security compliance is increasingly critical in HPC, and embedding it into engineering practices is key to long-term success. Our approach demonstrates that ReFrame, a tool already familiar to HPC teams, can serve as a practical and extensible platform for compliance validation. By integrating with ComplianceAsCode and adopting modern software engineering principles, we reduce the overhead of compliance while improving the overall security posture of our systems. Future efforts will focus on increasing portability, coverage, and community collaboration.

ACKNOWLEDGMENTS

The authors thank the CSCS engineers for their support and input during the development of this library, especially Gianni Riccardi and Leandro Machado. The ReFrame development team at CSCS and the ReFrame community are acknowledged for their continuous support of the framework. The authors acknowledge the use of ChatGPT (https://chatgpt.com) to improve sentences in the text in all sections.

REFERENCES

[1] Ansible Project Contributors. 2025. Ansible Documentation. https://docs.ansible.com. Accessed: 2025-02-30.

[2] ANSSI. 2022. Configuration recommendations of a GNU/Linux system. https://cyber.gouv.fr/publications/configuration-recommendations-gnulinux-system. Accessed: 2025-02-30.
[3] Center for Internet Security. 2025. CIS Benchmarks. https://www.cisecurity.org/cis-benchmarks. Accessed: 2025-02-30.
[4] Mike Chapple, James Michael Stewart, and Darril Gibson. 2018. *(ISC)2 CISSP Certified Information Systems Security Professional Official Study Guide* (8th ed.). SYBEX Inc., USA.
[5] Chef Software, Inc. 2025. Chef: Infrastructure Automation. https://www.chef.io. Accessed: 2025-02-30.
[6] CISOfy. 2025. Lynis: Security Auditing Tool for UNIX-based Systems. https://github.com/CISOfy/lynis. Accessed: 2025-02-30.
[7] ComplianceAsCode Contributors. 2025. ComplianceAsCode Content. https://github.com/ComplianceAsCode/content. Accessed: 2025-02-30.
[8] Peter F. Edemekong, Pavan Annamaraju, Muriam Afzal, and Micelle J. Haydel. 2024. Health Insurance Portability and Accountability Act (HIPAA) Compliance. https://www.ncbi.nlm.nih.gov/books/NBK500019/. Accessed: 2025-02-30.
[9] Florian Roth. 2025. Loki: Simple IOC and YARA Scanner. https://github.com/Neo23x0/Loki. Accessed: 2025-02-30.
[10] GNU Project. 2025. Bash - GNU Project - Free Software Foundation. https://www.gnu.org/software/bash/. Accessed: 2025-04-09.
[11] Goss Contributors. 2025. Goss: A Simple, Fast, and Lightweight Open Source Test Framework for Infrastructure. https://github.com/goss-org/goss. Accessed: 2025-02-30.
[12] International Organization for Standardization. 2025. ISO - International Organization for Standardization. https://www.iso.org. Accessed: 2025-02-30.
[13] ISO 27001:2022(EN) 2022. *Information security, cybersecurity and privacy protection — Information security management systems*. Standard. International Organization for Standardization, Geneva, CH.
[14] Harold Booth (NIST) Karen Scarfone (Scarfone Cybersecurity) Dragos Prisaca (G2) JDavid Waltermire (NIST), Stephen Quinn (NIST). 2018. *The Technical Specification for the Security Content Automation Protocol (SCAP): SCAP Version 1.3.* Technical Report NIST Special Publication (SP) 800-126 Rev. 3. National Institute of Standards and Technology, Gaithersburg, MD. https://doi.org/10.6028/NIST.SP.800-126r3
[15] Vasileios Karakasis, Theofilos Manitaras, Victor Holanda Rusu, Rafael Sarmiento-Pérez, Christopher Bignamini, Matthias Kraushaar, Andreas Jocksch, Samuel Omlin, Guilherme Peretti-Pezzi, João P. S. C. Augusto, Brian Friesen, Yun He, Lisa Gerhardt, Brandon Cook, Zhi-Qiang You, Samuel Khuvis, and Karen Tomko. 2020. Enabling Continuous Testing of HPC Systems Using ReFrame. In *Tools and Techniques for High Performance Computing*, Guido Juckeland and Sunita Chandrasekaran (Eds.). Springer International Publishing, Cham, 49–68.
[16] National Institute of Standards and Technology. 2025. NIST - National Institute of Standards and Technology. https://www.nist.gov. Accessed: 2025-02-30.
[17] Nextron Systems GmbH. 2025. THOR Lite - Nextron Systems. https://www.nextron-systems.com/thor-lite/. Accessed: 2025-02-30.
[18] OpenSCAP Project. 2025. Security Compliance - OpenSCAP. https://www.open-scap.org/features/security-compliance/. Accessed: 2025-02-30.
[19] Pavilion 2 Contributors. 2025. Pavilion 2 Documentation. https://pavilion2.readthedocs.io/en/latest/. Accessed: 2025-02-30.
[20] PCI Security Standards Council. 2025. PCI Security Standards Council. https://www.pcisecuritystandards.org. Accessed: 2025-02-30.
[21] Puppet, Inc. 2025. Puppet: IT Automation and Orchestration. https://www.puppet.com. Accessed: 2025-02-30.
[22] pytest Contributors. 2025. Pytest Documentation. https://docs.pytest.org. Accessed: 2025-02-30.
[23] RSpec Contributors. 2025. RSpec: Behaviour-Driven Development for Ruby. https://rspec.info. Accessed: 2025-02-30.
[24] Ruby Contributors. 2025. Ruby Programming Language. https://www.ruby-lang.org. Accessed: 2025-02-30.
[25] Serverspec Project Contributors. 2025. Serverspec: RSpec Tests for Your Servers. https://serverspec.org. Accessed: 2025-02-30.
[26] Testinfra Contributors. 2025. Testinfra Documentation. https://testinfra.readthedocs.io/en/latest/index.html. Accessed: 2025-02-30.
[27] U.S. Department of Defense. 2025. Security Technical Implementation Guides (STIGs). https://public.cyber.mil/stigs/. Accessed: 2025-02-30.
[28] YAML Core Team. 2025. YAML Ain't Markup Language (YAML). https://yaml.org. Accessed: 2025-02-30.