# Utilizing Test Anything Protocol (TAP) for Compliance Controls Evidence in SDLC and HITRUST

This document outlines a strategy for integrating the Test Anything Protocol (TAP) in managing compliance controls, specifically in the context of SDLC and HITRUST requirements. The focus is on differentiating between machine-generated and human attestations for compliance evidence, and leveraging TAP as a tool to streamline and enhance the reliability of compliance tracking, especially for controls requiring human attestations.

Below are screenshots of some types of reports that a TAP parser can produce; note that the content of TAP is plaintext and can be prepared by humans or machines and the output can be in text, graphical, or HTML formats.

A computer screen with text

Description automatically generated

A screenshot of a computer

Description automatically generated

**Introduction**

In the landscape of SDLC and HITRUST compliance, evidence collection is pivotal. Compliance controls evidence can be broadly classified into two categories: machine-generated attestations and human attestations. The integration of TAP offers an innovative approach to managing both, with a particular emphasis on the challenges associated with human attestations.

**Machine-Generated Attestations**

* **Definition**: Evidence generated automatically by systems or tools.
* **Characteristics**: Highly reliable, time-efficient, easily reproducible.
* **Examples**: System logs, automatic audit trails.

**Human Attestations**

* **Definition**: Evidence that requires human intervention, judgment, or decision-making.
* **Characteristics**: Prone to subjectivity, requires more time, often not easily reproducible.
* **Examples**: Manual review approvals, physical security checks.

**Test Anything Protocol (TAP) Overview**

TAP, originating from the Perl community, is a simple text-based interface protocol used for testing output. Its flexibility and language-agnostic nature make it a suitable candidate for compliance evidence management.

**TAP for Machine Attestations**

* **Application**: Automated test scripts producing TAP output for system-level verifications.
* **Advantage**: Creates a standardized output format for machine-generated evidence.

**TAP for Human Attestations**

* **Application**: Structuring human-performed checks or audits in a TAP format.
* **Advantage**: Provides a consistent framework for recording and reviewing human-generated compliance evidence.

**Strategy for Implementing TAP in Compliance Controls**

**Creating TAP Files for Compliance Evidence**

1. **For Machine-Generated Attestations**:
   * Develop automated scripts that perform compliance checks and generate TAP output.
   * Example: A script that verifies data encryption standards and outputs TAP results.
2. **For Human Attestations**:
   * Design templates for manual compliance tasks in a TAP format.
   * Implement a simple interface for human operators to input TAP results.
   * Example: A TAP template for physical access control checks.

**Storing TAP Files in Git Repositories**

* **Version Control**: Utilize Git for version controlling TAP files, ensuring an auditable history of compliance evidence.
* **Accessibility**: Facilitates easy access and review of compliance data for auditors.
* **Security**: Leverages Git's inherent security features for sensitive compliance data.

**Significance of TAP in Human Attestations**

* **Standardization**: Brings a level of standardization and structure to subjective human attestations.
* **Transparency**: Enhances transparency in how human-based compliance checks are conducted and recorded.
* **Audit Trail**: Offers a clear, chronological trail of human actions and decisions.

**Implementation Plan**

1. **Assessment & Planning**: Evaluate current compliance controls and identify areas for TAP integration.
2. **Development & Integration**: Develop TAP templates and scripts, and integrate them with existing systems and procedures.
3. **Training**: Conduct training sessions for staff to familiarize them with TAP-based evidence recording.
4. **Testing & Refinement**: Pilot test the TAP implementation and refine based on feedback.
5. **Deployment**: Full-scale deployment of TAP for compliance controls evidence management.
6. **Monitoring & Review**: Continuous monitoring and periodic review of the TAP implementation.

The implementation of the Test Anything Protocol (TAP) for managing compliance controls evidence, particularly in validating your SDLC and considering HITRUST frameworks, presents a significant opportunity to enhance the reliability and efficiency of compliance tracking. While beneficial for both machine-generated and human attestations, TAP's role in standardizing and structuring human attestations is particularly transformative, offering a new level of rigor and transparency in compliance management.

# Utilizing TAP for SDLC Controls Evidence Preparation

The Test Anything Protocol (TAP) can be effectively utilized to prepare evidence for various Software Development Life Cycle (SDLC) controls. Below are examples and explanations of how TAP can be applied to common SDLC controls such as ensuring functional specifications and requirements are completed, documenting architecture and strategy decision documents, conducting peer reviews, and verifying that tests align with expected requirements.

**1. Evidence for Functional Specifications and Requirements Completion**

TAP Application:

* **Creation of TAP Reports**: After the completion of functional specifications and requirements, a TAP report can be generated to document the completion status of each requirement.
* **Example**: A TAP report may list each requirement with a status of 'ok' or 'not ok', along with comments detailing the completion status or pending actions.

Explanation:

* This approach ensures that each requirement is accounted for and provides a clear, timestamped record of the completion status, enhancing traceability and accountability.

**2. TAP Evidence for Architecture and Strategy Decision Documents**

TAP Application:

* **Document Review Tracking**: TAP can be used to track the review and approval process of architecture and strategy documents.
* **Example**: A TAP report could include entries for each review step, indicating whether the document meets the set criteria (e.g., 'ok' for approval, 'not ok' for revisions required).

Explanation:

* Utilizing TAP in this manner ensures a documented, auditable trail of the decision-making process, highlighting compliance with established SDLC protocols.

**3. TAP Evidence of Peer Reviews (Design Reviews and Code)**

TAP Application:

* **Peer Review Logs**: Implement TAP to log outcomes of peer reviews for design and code.
* **Example**: For each peer review session, a TAP report can be generated indicating the success ('ok') or issues ('not ok') found during the review, along with reviewer comments.

Explanation:

* This systematizes the peer review process, providing clear and structured evidence of review outcomes and actions taken, which is crucial for quality assurance and compliance.

**4. TAP as a Mechanism for Proving Tests Against Expected Requirements**

TAP Application:

* **Test Case Mapping**: Use TAP to map test cases to specific requirements, documenting the test results.
* **Example**: A TAP report can list each test case, its corresponding requirement, and the test outcome, providing a direct link between tests conducted and requirements.

Explanation:

* This method clearly demonstrates the coverage and effectiveness of testing in relation to the defined requirements, a critical aspect of the SDLC process.

**Storing TAP Reports in Git Repositories**

* **Version Control**: Use Git to maintain versions of TAP reports, ensuring an up-to-date and historical record of the SDLC process.
* **Collaboration and Accessibility**: Git repositories facilitate collaborative review and easy access to TAP reports for all stakeholders.

TAP's versatility makes it an excellent tool for documenting and validating various stages of the SDLC process. By providing structured, verifiable, and easily accessible evidence for key SDLC controls, TAP enhances the transparency, accountability, and compliance of the software development process. Integrating TAP into SDLC controls not only aids in meeting compliance requirements but also contributes to the overall quality and reliability of software products.

Controls Compliance Examples

## Functional Specifications and Requirements Completion

TAP version 13

1..4

ok 1 - SCF Control ID: SYS.01 - Requirement #1234 completed

- Note: All criteria met, detailed in Spec\_Doc\_v1.2.pdf

- GitHub Issue: #456

ok 2 - SCF Control ID: SYS.02 - Requirement #1235 completed

- Note: Pending minor revisions. See comments in Jira ticket ABC-123

- Jira Ticket: ABC-123

not ok 3 - SCF Control ID: SYS.03 - Requirement #1236 incomplete

- Note: Awaiting approval. PR at https://github.com/repo/pull/789

- Pull Request: https://github.com/repo/pull/789

ok 4 - SCF Control ID: SYS.04 - Requirement #1237 completed

- Note: Reviewed and approved in meeting on 03/05/2024. Minutes in Meeting\_Minutes\_030524.docx

- Related Document: Meeting\_Minutes\_030524.docx

## Architecture and Strategy Decision Documents

TAP version 13

1..3

ok 1 - SCF Control ID: GOV.01 - Architecture document reviewed

- Note: Approved in Architecture\_Review\_Meeting\_030624. Document: Architecture\_v2.3.pdf

- Meeting Minutes: Architecture\_Review\_Meeting\_030624.docx

not ok 2 - SCF Control ID: GOV.02 - Strategy document pending revisions

- Note: Requires updates as per feedback in Jira ticket XYZ-789

- Jira Ticket: XYZ-789

ok 3 - SCF Control ID: GOV.03 - Security strategy approved

- Note: Security\_Strategy\_v1.0.pdf approved. Refer to Pull Request for implementation details

- Pull Request: https://github.com/repo/pull/1011

## Peer Reviews (Design Reviews and Code)

TAP version 13

1..2

ok 1 - SCF Control ID: DEV.01 - Design review for Feature A

- Note: Design approved with minor suggestions. File: FeatureA\_Design.docx

- Reviewed File: FeatureA\_Design.docx

- GitHub Issue: #789

not ok 2 - SCF Control ID: DEV.02 - Code review for Module X

- Note: Several issues identified. Refer to comments in Pull Request

- Pull Request: https://github.com/repo/pull/1213

## Tests Against Expected Requirements

TAP version 13

1..3

ok 1 - SCF Control ID: QA.01 - Test Case TC001 for Requirement #1234

- Note: Passed. Validates user authentication flow.

- Related Requirement Doc: User\_Auth\_Spec\_v1.1.pdf

ok 2 - SCF Control ID: QA.02 - Test Case TC002 for Requirement #1235

- Note: Failed. Issue tracked in GitHub Issue #4567

- GitHub Issue: #4567

ok 3 - SCF Control ID: QA.03 - Test Case TC003 for Requirement #1237

- Note: Passed. Checks data encryption standards.

- Test Script: EncryptTest\_v2.js

Audit Report Examples

If a SOC2 or HITRUST auditor were to document their audit findings in a TAP format, their reports would likely include structured, concise entries for each control examined, indicating the compliance status and any relevant observations or recommendations. Here are some examples of what such audit reports might look like in TAP format:

# Example 1: HITRUST Audit Report

TAP version 14

1..X

# HITRUST e1 Assessment CSF Requirements - Expanded Domains and BUIDs with Diagnostic Information

# Example Domain: Endpoint Security

ok BUID XXXX.XXm1: Endpoint security policy compliance

---

evidence: "Policy compliance report from ComplianceChecker v1.2"

cli\_output: "EndpointPolicyCheck --date 2023-09-15 --result pass"

policy\_document: "EndpointSecurityPolicy\_v3.pdf"

jira\_issue: "SEC-789 - Annual policy review and update"

...

# Example Domain: Data Protection

ok BUID XXXX.XXz1: Encryption of sensitive data at rest

---

evidence: "Encryption status report generated by DataProtectTool v5.4"

script\_output: "DataEncryptStatus --scope all --date 2023-09-20"

github\_issue: "https://github.com/org/datasecurity/issues/112 - Encryption verification"

...

# Example Domain: Incident Response

ok BUID XXXX.XXc1: Incident response plan testing

---

evidence: "IR plan test scenario and outcomes document"

test\_date: "2023-08-30"

test\_scenario: "Simulated data breach response"

jira\_issue: "IR-203 - Post-test review and action items"

...

# Example Domain: Vendor Management

ok BUID XXXX.XXi1: Vendor security assessment completion

---

evidence: "Vendor assessment report summary"

assessment\_tool: "VendorRiskAssessment v2.1"

assessment\_date: "2023-07-05"

jira\_issue: "VM-457 - Assessment findings and remediation tracking"

...

In this example:

* **ok N**: Each line begins with either **ok** or **not ok** to indicate whether the specific requirement (identified by a BUID) has been met. **N** represents a sequential test number.
* **evidence** and other YAML block entries: Provide specific details on the type of evidence supporting compliance, such as direct output from compliance or security tools (**cli\_output**), references to policy documents, issue tracking IDs (**jira\_issue**, **github\_issue**), and details about the evidence collection or verification process (**script\_output**, **test\_date**, etc.).

This format ensures each compliance assertion within the HITRUST assessment is supported by clear, actionable evidence. It's crucial to tailor the evidentiary support to the specific controls and processes within your organization, leveraging the documentation, tool outputs, and issue tracking systems you use to manage and verify compliance.

# Example 2: SOC2 Audit Report

TAP version 14

1..X

# SOC2 Assessment - Trust Services Criteria with Diagnostic Information

# Domain: Security

ok Control ID S1: Network firewall configuration and effectiveness

---

evidence: "Firewall configuration audit report"

cli\_output: "FirewallAuditTool --date 2023-09-15 --status pass"

policy\_document: "NetworkSecurityPolicy\_v2.pdf"

jira\_issue: "NETSEC-101 - Firewall configuration review"

...

ok Control ID S2: Data encryption for sensitive information

---

evidence: "Data encryption verification report"

script\_output: "EncryptCheck --scope sensitive\_data --result all\_encrypted"

github\_issue: "https://github.com/org/security/issues/204 - Encryption policy enforcement"

...

# Domain: Availability

ok Control ID A1: Backup and recovery plan testing

---

evidence: "Disaster recovery plan test results"

test\_date: "2023-08-01"

test\_scenario: "Recovery from backup simulation"

jira\_issue: "DR-312 - DR test follow-up actions"

...

# Domain: Processing Integrity

ok Control ID PI1: Transaction processing monitoring and validation

---

evidence: "Transaction integrity audit logs"

monitoring\_tool: "TransactMonitor v3.5"

review\_period: "2023 Q3"

jira\_issue: "TXN-401 - Quarterly transaction review"

...

# Domain: Confidentiality

ok Control ID C1: Access control to confidential data

---

evidence: "Access control audit report"

audit\_tool: "AccessAudit v4.2"

audit\_date: "2023-07-15"

jira\_issue: "DATA-509 - Access control improvements"

...

# Domain: Privacy

ok Control ID P1: Personal data processing and consent management

---

evidence: "Privacy compliance review document"

review\_date: "2023-06-20"

policy\_document: "PrivacyPolicy\_v3.1.pdf"

github\_issue: "https://github.com/org/privacy/issues/58 - Consent mechanism update"

...

In this SOC2 example:

* **ok N**: Each line starts with **ok** to indicate compliance with a SOC2 control, followed by a control ID (e.g., S1, A1) for identification.
* **Diagnostic information**: Includes detailed evidence supporting compliance, such as audit reports, policy documents, direct output from monitoring or audit tools, and references to issues where findings and remediations are tracked.

This structured approach ensures each SOC2 control's compliance status is clearly communicated, backed by specific, actionable evidence. Tailoring the evidentiary support to fit your organization's processes and the systems used to manage SOC2 compliance is essential for creating a meaningful and auditable TAP output.

# Example 2: SCF Audit Report

TAP version 14

# SCF Compliance Assessment - Sample TAP Output with Diagnostic Information

# Control Domain: Asset Management

ok - SCF Control AM-1: Inventory of Assets

---

evidence: "Automated asset discovery report via AssetDiscoveryTool v3.2"

script\_output: "AssetDiscoveryTool --run-date 2023-09-10"

jira\_issue: "AM-101 - Review and update asset inventory list"

...

# Control Domain: Access Control

ok - SCF Control AC-2: Account Management

---

evidence: "User account review completed"

review\_tool: "UserReviewTool v2.0 --output summary"

policy\_document: "AccessControlPolicy\_v5.pdf"

github\_issue: "https://github.com/org/security/issues/321 - User account audit actions"

...

ok - SCF Control AC-6: Least Privilege

---

evidence: "Privilege access audit log"

audit\_period: "2023 Q2"

audit\_tool\_output: "PrivilegeAudit v1.5 --result compliant"

jira\_issue: "AC-204 - Remediation of excessive permissions identified"

...

# Control Domain: Data Security

ok - SCF Control DS-2: Data Encryption

---

evidence: "Encryption status check for stored data"

cli\_output: "EncryptStatusChecker --target all\_stored\_data --date 2023-09-15"

policy\_compliance: "Compliant with DataSecurityPolicy\_v4.3"

jira\_issue: "DS-307 - Encryption policy compliance verification"

...

# Control Domain: Information Protection Processes and Procedures

ok - SCF Control IP-8: Disposal of Sensitive Information

---

evidence: "Data disposal and destruction record"

disposal\_date: "2023-08-22"

method: "Secure shredding and electronic wipe verified"

jira\_issue: "IP-415 - Secure disposal of project X data"

...

# Control Domain: Risk Assessment

ok - SCF Control RA-1: Risk Assessment Policy and Procedures

---

evidence: "Annual risk assessment report"

assessment\_date: "2023-07-30"

risk\_tool: "RiskAssessmentPro v3.7"

github\_issue: "https://github.com/org/risk-team/issues/112 - Risk assessment findings and next steps"

...

In this SCF example:

* **ok**: Indicates compliance with each SCF control.
* **Diagnostic information**: Provides detailed evidence for compliance, including automated tool reports, policy document references, and links to issue tracking for human-attested evidence or remediation actions.

This TAP output format ensures that compliance with SCF controls is clearly documented, supported by specific evidence that can be traced back to policy documents, tool outputs, or documented issues, facilitating a comprehensive and auditable compliance process.

# Key Points in TAP Audit Reports:

* **TAP Version**: Indicates the version of TAP used for the report.
* **Control Count**: Specifies the number of controls audited.
* **Control Status (ok or not ok)**: Indicates whether each control meets the compliance standards.
* **SCF Control ID**: References the specific control ID from the Secure Controls Framework.
* **Note**: Provides observations or specific findings related to the control.
* **Recommendation**: Includes suggestions for improvement or corrective actions where non-compliance is found.

Using TAP for audit reports could provide a clear, structured, and easy-to-understand format for both auditors and the organizations being audited, facilitating a straightforward assessment of compliance status and required actions.Top of Form

SDLC (“Quality System”) TAP File Storage Specification

**Introduction**

This specification defines the standard for storing TAP (Test Anything Protocol) files in source control repositories for quality systems (SDLC). It is modeled after the XDG Base Directory Specification, tailored to meet the needs of managing TAP files effectively within the context of software development and compliance auditing.

**Basics**

The Quality System TAP File Storage Specification is founded on these concepts:

1. **Root Directory for TAP Files**: There is a single base directory relative to which TAP files should be stored in source control repositories. This directory is **.quality-system**.
2. **Environment Variable Prefix**: The prefix for environment variables used in this specification is **QS**.
3. **Preference-Ordered Search**: There is a set of preference-ordered base directories relative to which TAP files should be searched within the **.quality-system** directory.
4. **Absolute Path Requirement**: All paths set in these environment variables must be absolute. Relative paths are considered invalid and ignored.

**Environment Variables**

1. **QS\_TAP\_HOME**:
   * Defines the base directory relative to which user-specific TAP files should be stored.
   * Default: If **QS\_TAP\_HOME** is not set or empty, the default is **$REPO\_HOME/.quality-system/assurance**.
2. **QS\_TAP\_DIRS**:
   * Defines the preference-ordered set of base directories to search for TAP files in addition to the **QS\_TAP\_HOME** base directory.
   * Directories in **QS\_TAP\_DIRS** should be separated with a colon ':'.
   * Default: If **QS\_TAP\_DIRS** is not set or empty, the default is $**QS\_TAP\_HOME for engineering contributors**. **/usr/local/share/quality-system/assurance:/usr/share/quality-system/assuranceare the defaults for system / machine attestation**.
3. **QS\_TAP\_CONFIG**:
   * Defines the base directory relative to which TAP configuration files should be stored.
   * Default: If **QS\_TAP\_CONFIG** is not set or empty, the default is **$ REPO\_HOME/.quality-system/conf**.
4. **QS\_TAP\_CACHE**:
   * Defines the base directory relative to which user-specific non-essential TAP data should be stored.
   * Default: If **QS\_TAP\_CACHE** is not set or empty, the default is **$ REPO\_HOME/.quality-system/cache**.

**Directory Structure**

* The **.quality-system** root directory should contain subdirectories for different types of TAP files, such as:
  + **/assurance**: For storing TAP and other compliance files.
  + **/config**: For configuration files related to TAP processing.
  + **/cache**: For non-essential, cacheable TAP data.

**File Naming Conventions**

* TAP files should be named meaningfully, reflecting their content. For example, **audit-report-2024-01-26.tap** for an audit report generated on January 26, 2024.

**Referencing the Specification**

* Other specifications or documents may reference this specification by specifying the location of a TAP file as **$QS\_TAP\_DIRS/subdir/filename**.
* This implies that the file should be searched for in **./subdir/filename** relative to all base directories specified by **QS\_TAP\_HOME** and **QS\_TAP\_DIRS**.
* If an environment variable is not set or is empty, its default value as defined by this specification should be used.

**File Access and Error Handling**

* When writing a TAP file, if the destination directory is non-existent, an attempt should be made to create it with permission 0700. Existing directory permissions should not be altered.
* When reading a TAP file, if a file in a certain directory is inaccessible (due to non-existence, lack of authorization, etc.), the processing of the file in that directory should be skipped. Appropriate error messages may be presented to the user if necessary.

This SDLC (Quality System) TAP File Storage Specification aims to standardize the storage and retrieval of TAP files in source control repositories, ensuring a structured, efficient, and scalable approach to managing test and audit data within quality systems.