Why are we all here today?

- Information technology is complex & calls for automation
- Regulatory frameworks are burdensome & Need interop auto GRC tools
- Paper-based A&A doesn’t scale & Calls for auto updates
- Security vulnerabilities are everywhere & Calls for auto updates
- Risk management is hard & Experts need automation
- DevOps & IaC is hard in multi-clouds & Calls for interoperability & standardization
What was needed?

A (Cyber) Machine-readable Esperanto that enables actors, tools and organizations to exchange information via automation:

OSCAL sets the foundation for automation and interoperability
OSCAL provides a common/single machine-readable language, expressed in XML, JSON and YAML for:

- multiple compliance and risk management frameworks (e.g. SP 800-53, ISO/IEC 27001&2, COBIT 5)
- software and service providers to express implementation guidance against security controls (Component definition)
- sharing how security controls are implemented (System Security Plans [SSPs])
- sharing security assessment plans (System Assessment Plans [SAPs])
- sharing security assessment results/reports (System Assessment Results [SARs])

OSCAL enables automated traceability from selection of security controls through implementation and assessment.
How is OSCAL different?

- No information needs duplication
- Custom granularity (controls can be decomposed into statements)
- Unique identifiers for parameters and statements
- Vendors can document their products
- Systems’ security implementation can be decomposed
- Capture assessment Plans and Activities with custom cadence, & only for selected components
- POA&M conveys open risks aligned with the SSP capabilities and controls
Anatomy of the OSCAL Implementation Layer
OSCAL SSP:

- Imports a Profile identifying the controls
- Each control response is broken down to the individual components involved.
- Enables a more robust response to controls
- Example: The access control implementation that satisfies AC-2, part a is described separately for:
  - This System
  - The Access Control Procedure
  - A shared Application

*Every SSP must have a component representing the whole system.*
OSCAL Support for Continuously Authorizing Systems to Operate

Authorization to Use Common Control Authorization
## Common Control Authorization & Authorization to Use

### Yes

- **Customer Org.**
  - Cust 1
  - Cust 2
  - Cust 3
  - Cust 4
  - Cust 5
  - Cust 6

- **Leveraging System**
  - Leveraging SaaS A
  - Leveraging SaaS B
  - Leveraging SaaS C

- **Authorization to Use**
  - Leveraged IaaS

**Cloud**: Several SaaS systems running on a separately authorized IaaS.

### Yes

- **Customer Org.**
  - Cust 1
  - Cust 2
  - Cust 3
  - Cust 4
  - Cust 5
  - Cust 6

- **Leveraging System**
  - System A (Application)
  - System B (Application)

- **General Support System**
  - Active Directory w/SSO
  - Storage Area Network
  - Network Infrastructure

**Data Center**: Several systems relying on a separately authorized storage array or other general support system (GSS)

### No

- **Customer Org.**
  - Cust 1
  - Cust 2
  - Cust 3
  - Cust 4

- **Leveraging System**
  - Identity Management Service
  - Leveraging SaaS A
  - Leveraging SaaS B

- **General Support System**
  - IaaS
  - IaaS

**External Service or Interconnection**

- Interconnections or External Services are not leveraged authorizations
- Even if they have an authorization
- SaaS A handles the Identity Management Service as a system component
- OSCAL supports this, just not as a L.A.
Leveraging System:
The leveraging system's SSP should:
- identify what is inherited from a leveraged system
- identify any addressed responsibilities (as identified by the leveraged system)

In addition to:
- identifying what may be inherited by the leveraging system's customers
- any responsibilities the leveraging system's customers must address to fully satisfy a control

OSCAL supports leveraged ATOs of complex stacked systems
The same syntax is used
- It is simply replicated for each leveraged system

The Leveraging System's SSP:
- Has a separate "leveraged-authorization" assembly for each leveraged system.
- Has a separate "component" representing each leveraged system.
- Has a separate "component" representing the leveraged system components associated with inherited capabilities.
Anatomy of the OSCAL Assessment Layer
Assessment Plan (AP) & Assessment Results (AR)

- OVERLAPPING SYNTAX
- SIMILAR BUT DISTINCT PURPOSE
- UNIQUE to AR: Results (& Evidence)

Continuous Assessment Approach
- Assessment Plan: What should be tested/inspected, how, and with which cadence is desired
- Assessment Results: Time-slice of results
OSCAL POA&M Model

System Security Plan (SSP)
- Metadata
  - role, party (person/org/team)
- System Characteristics
- System Implementation
  - Leveraged Authorization
  - User
  - Component [This System] *
    - Component (Access Control Process)
    - Component (Linux OS)
- System Inventory
  - Inventory Item
- Control Implementation
  - Implemented Requirement (AC-1)
  - Implemented Requirement (AC-2)
  - Implemented Requirement (AC-3)
- Back Matter
  - Attachments and Citations

Assessment Results (AR)
- Import Assessment Plan
- Local Definitions
- Results (Current)
  - Local Definitions
  - Reviewed Controls
  - Attestations
  - Assessment Log
- Observations, Risks & Findings
  - Identified Risks, Calculations Deviations
  - Recommendations and Remediation Plans
  - Evidence Descriptions and Links
  - Disposition Status
- Results (Last Cycle)
- Results (Earlier Cycle)

Plan of Action and Milestones (POA&M)
- Metadata
  - Title, Version, Date
  - Roles, People, Organizations
- Import SSP
  - Pointer to FedRAMP System Security Plan
- System Identifier
  - Unique system ID
- Local Definitions
  - Observations, Risks
- POA&M Items
  - Unique ID, Impacted Control
- Observations
- Risk Information
  - Title, Source, CVE#, Severity
- Remediation Activities
  - Plan, Schedule, Resolution Date, Remediation Status
- Vendor Dependencies
  - Evidence and Check-Ins
- Deviations
  - Status (Investigating, Pending, Approved)
  - False Positive (FP)
- Operational Requirement (OR)
- Risk Adjustment (RA)
- CVSS Metrics
- Back Matter
  - Citations and External Links
  - Attachments and Embedded Images
  - Evidence (Vendor Check-Ins, DR Evidence)
What can you do with the OSCAL models?
OSCAL Models    >>>    OSCAL Content    >>>   OSCAL Tools

OSCAL Models

https://github.com/usnistgov/OSCAL

OSCAL Content Generation

https://github.com/usnistgov/oscal-content

OSCAL Editorial Tools

https://github.com/usnistgov/oscal-tools

OSCAL GRC Tools

https://github.com/usnistgov/oscal-tools
Who can benefit & How

Risk Management & OSCAL content

NIST

RMF steps: PREPARE  CATEGORIZE  SELECT  IMPLEMENT  ASSESS  AUTHORIZE  CON-MON
Use of Metaschema and Metaschema Constraints in OSCAL
Metaschema provides a format-agnostic foundation for managing data formats and supporting tooling

- OSCAL uses Metaschema to define the logical structure of OSCAL data.

- Metaschema is format agnostic. We can support XML, JSON, and YAML with no extra effort.

- XML and JSON Schema are derived from Metaschema definitions to support well-formedness checking for XML, JSON, and YAML formats.

- Model documentation can be auto-generated.

Metaschema provides a format-agnostic foundation for managing the OSCAL models.
A single modeling approach supporting multiple formats and related productions.
- Metaschema is compositional.

- **Imports** allow constructs to be shared between Metaschemas.

- **Assembly**: Represents a data object.
  - Contains child assemblies and fields at various cardinalities
  - May contain flags

- **Field**: Represents a valued data point
  - May contain flags

- **Flag**: Qualifies or supports a data object
The path to richer Semantics in OSCAL

- Metaschema supports format-agnostic constraints.
  - Supports co-constrains (i.e., if X then Y)
  - Richer rules than schema validation alone

- Constraints are written in Metapath, an XPath-like grammar.

- Can measure data completeness for a given use case

- Will allow different dialects to be defined for OSCAL (i.e., FedRAMP)
## Open-Source Tools and Libraries


<table>
<thead>
<tr>
<th>Name</th>
<th>Provider/Developer</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compliance trestle</td>
<td>IBM</td>
<td>A python SDK and command line tool which manipulates OSCAL structures and supports transformation of data into OSCAL.</td>
<td>open source</td>
</tr>
<tr>
<td>OSCAL Java Library</td>
<td>NIST OSCAL Project</td>
<td>A Java-based programming API for reading and writing content conformant to the OSCAL XML, JSON, and YAML based models.</td>
<td>open source</td>
</tr>
<tr>
<td>OSCAL React Component Library</td>
<td>Easy Dynamics</td>
<td>A library of reusable React components and an example user interface application that provides a direct UI into OSCAL.</td>
<td>open source</td>
</tr>
<tr>
<td>OSCAL REST API</td>
<td>Easy Dynamics</td>
<td>An initial OpenAPI definition of an OSCAL REST API that describes how systems might manipulate catalogs, profiles, components, and SSPs.</td>
<td>open source</td>
</tr>
<tr>
<td>XSLT Tooling</td>
<td>NIST OSCAL Project</td>
<td>A variety of Extensible Stylesheet Language (XSL) Transformations (XSLT), Cascading Style Sheets (CSS), and related utilities for authoring, converting, and publishing OSCAL content in various forms.</td>
<td>open source</td>
</tr>
<tr>
<td>XML Jelly Sandwich</td>
<td>Wendell Piez (NIST)</td>
<td>Interactive XSLT in the browser includes OSCAL demonstrations.</td>
<td>open source</td>
</tr>
<tr>
<td>Xacta 360</td>
<td>Telos</td>
<td>Xacta 360 is a cyber risk management and compliance analytics platform that enables users to create and submit FedRAMP system security plans (SSPs) in OSCAL format. Future OSCAL capabilities are forthcoming as the standard evolves.</td>
<td>license</td>
</tr>
<tr>
<td>Atlassity: Continuous Compliance Automation</td>
<td>C2 Labs</td>
<td>Atlassity CE (release 2.0) runs in any environment and supports the development of OSCAL v1.0 content for Catalogs, Profiles, System Security Plans and Components. Additional detail can be found in this blog post: Atlassity Delivers Free Tools to Create OSCAL Content.</td>
<td>community edition</td>
</tr>
<tr>
<td>control_freak</td>
<td>Risk Redux</td>
<td>This tool seeks to provide folks with a searchable and easy-to-navigate reference for NIST SP 800-53 Revision 5. It is an open-source application from the Risk Redux project, built using parsed content directly from the OSCAL repositories.</td>
<td>open-source</td>
</tr>
</tbody>
</table>
Few of the OSCAL Adopters

2021 presenters

- FedRAMP
- Noblis
- HHS CMS
- National Renewable Energy Lab
- GovReady
- C2 Labs
- cFocus Software
- Shujinko
- Robers Bosch (EU|Germany)
- Telos
- KPMG
- IBM Research

2022 new presenters

- AWS
- CSAM
- Easy Dymanics
- Volant Associates LLC
- Secureframe
- Red Hat
- Nirmata
- SunStone Secure

2021-2022 other adopters

- Booz Allen Hamilton
- eMASS
- Microsoft
- Coalfire
- Kratos
- Salesforce
- Oracle
OSCAL: the Open Security Controls Assessment Language

Automated Control-Based Assessment
Supporting Control-Based Risk Management with Standardized Formats

Providing control-related information in machine-readable formats.

NIST, in collaboration with industry, is developing the Open Security Controls Assessment Language (OSCAL). OSCAL is a set of formats expressed in XML, JSON, and YAML. These formats provide machine-readable representations of control catalogs, control baselines, system security plans, and assessment plans and results.
Publicly Available Resources

Documentation:
Catalog, Profile, Component, SSP, SAP, SAR, POA&M:
https://pages.nist.gov/OSCAL/documentation/

Example:
Generic examples:
https://github.com/usnistgov/oscal-content/tree/master/examples
NIST SP 800-53 R4 and Rev5 catalog and baselines (XML & JSON):
https://github.com/usnistgov/oscal-content/tree/master/nist.gov/SP800-53

FedRAMP Automation:
Repository (FedRAMP catalog and baselines (XML & JSON) included):
https://github.com/GSA/fedramp-automation

Tools
OSCAL Java Library: https://github.com/usnistgov/liboscal-java
XSLT Tooling: https://github.com/usnistgov/oscal-tools/tree/master/xslt
OSCAL Kit: https://github.com/docker/oscalkit
OSCAL GUI: https://github.com/brianrdiggs/OSCAL-GUI

Please visit Community’s:
OSCAL Club/awesome-oscal: https://github.com/oscal-club/awesome-oscal
Questions?

Contact us at: oscal@nist.gov
Chat with us on Gitter: https://gitter.im/usnistgov-OSCAL/Lobby
Collaborate with us on GitHub: https://github.com/usnistgov/OSCAL
Join our COI meetings: https://pages.nist.gov/OSCAL/contribute/#community-meetings

Thank you!