

From One-Size-Fits-All to Right-Sizing: Adapting OSCAL for the Singapore Government's Tech Standards

Hunter Nield, Distinguished Engineer

Eugene Lim, Lead Security Engineer



About Us



Hunter Nield

Distinguished Engineer, GovTech

Hunter is a Distinguished Engineer at the Government Technology Agency of Singapore, where he helps lead the Engineering and DevOps Centre of Excellence.



Eugene Lim

Lead Security Engineer, OGP

Eugene is a cybersecurity professional and white hat hacker who builds resilient appsec programmes in the day and researches product security at night.

GovTech and OGP



GovTech builds tech to improve the lives of Singaporeans and supports public agencies' efforts in driving digital transformation.



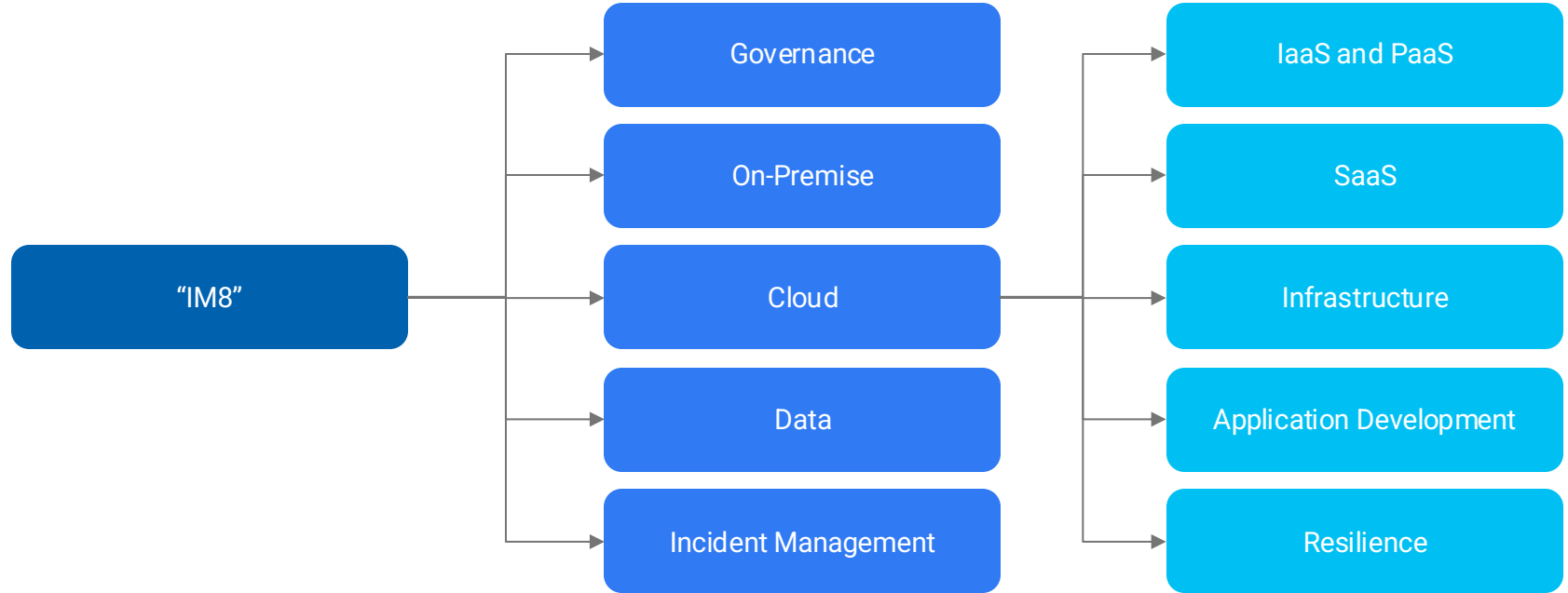
**OPEN
GOVERNMENT
PRODUCTS**

Open Government Products is an experimental development team that builds technology for the public good.

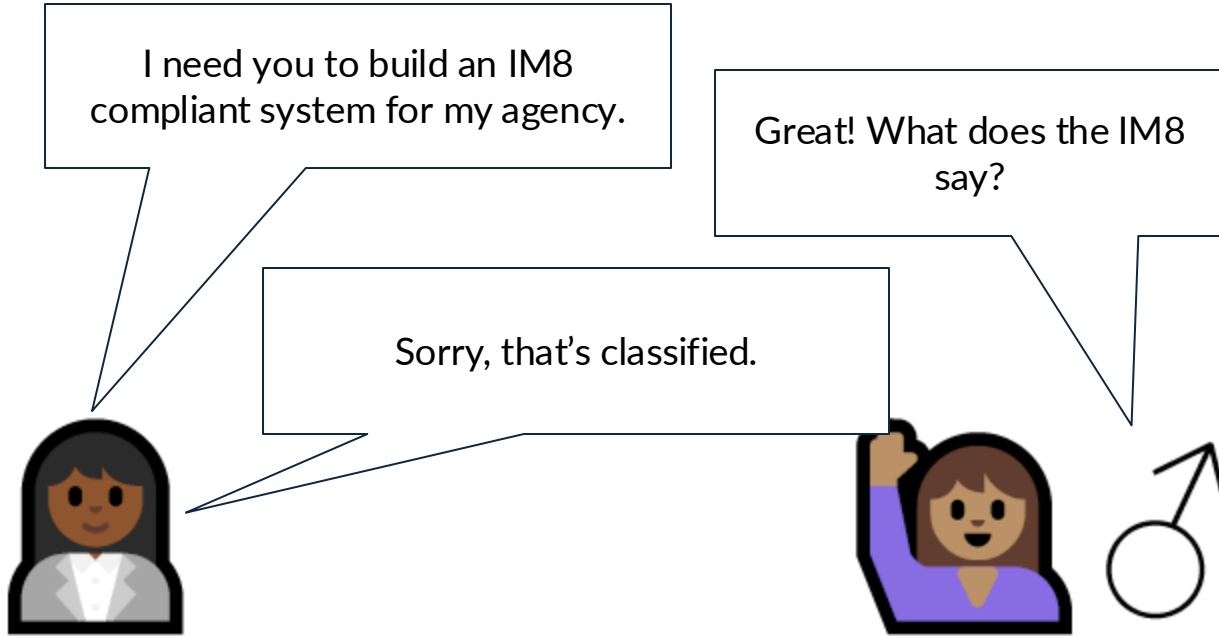
Challenges of One-Size-Fits All



The Instruction Manual for Infocomm Technology and Smart Systems (ICT&SS) began as internal policies, standards and guidelines that evolved organically and rapidly.



“IM8” began as a set of controls internal to the Singapore Government’s systems, rather than a vendor-facing compliance programme.



The one-size-fits-all monolithic format created structural problems, such as duplication and desynchronisation.

Cloud Application Development Control X

“Set up Static Application Security Testing to scan source code...”

On-Premise App Development Control Y

“Include the following capabilities in practising DevSecOps:

g) Static Application Security Testing (SAST) tools;”

Third Party Management Control Z

“The CI/CD pipeline shall minimally have:

b) Static Application Security testing tools;”

Lack of impact-tiering, implementing components, and system type distinctions created awkward control phrasing and design that impacted readability and usability.

“For systems classified CONFIDENTIAL and above, perform an account review every X days. For all other systems, perform an account review every Y days. For agencies that use an automated mechanism, review the notification rules and perform validation tests every Z days.”

“Implement the following 13 secure coding practices: ...”

Workarounds were created to express complexity, creating more complexity.

G: Control X##

denotes this is not applicable to SaaS

G denotes a guideline and is non-mandatory

Control Y** C10

** denotes this is not applicable for all cloud services

C10 denotes compliance by 1 Jan 2020

Control Z^{CX}

CX denotes compliance timelines that do not fall in the categories C1 to C11 and are defined separately

The OSCAL schema resolved many of these challenges.



Duplication

Common control catalog as a source of truth



Risk Tiering

Profiles to apply risk levels and must-/should-/good-to-have designation



System Types

System security plans and parameters to provide additional customisation

Customisations to our OSCAL approach



One change to OSCAL conventions came from a structural issue.



A “must-have” for a low-risk system may not be a “must-have” for a medium-risk system.



Since the risk materiality level of a system is linked to the level of enforcement, we couldn't use a single profile set.

Level 0

“Must Have”

No deviation allowed

Level 1

“Should Have”

Deviation allowed
after agency
committee approval

Level 2

“Good to Have”

Optional

This led to more verbosity at the profile level.



Moving from a one-size-fits-all compliance model to a risk-based, right-sized approach required bridging and change management.

I just want you to tell me what to do.

This is too much freedom!

I'm not equipped to make risk assessments.



We added a risk-statement property to guide agencies, and ran risk assessment workshops.

Statement: The requirements that must be implemented by the system.

Recommendations: Additional recommendations or information to consider when selecting, implementing, assessing, and monitoring a control. This is not a mandatory or audited requirement.

Risk: The security risk mitigated by the control.



We also distributed “template SSPs” with default parameter values set at the profile and SSP levels.

System Security Plans > Low Risk Cloud

Low-Risk Cloud

Overview

The Low-Risk Cloud System Security Plan template includes Level 0 and Level 1 baseline controls that are recommended as the default controls for low-risk cloud systems classified up to Restricted, Sensitive Normal systems, excluding National Emergency (NE) Critical, Critical Information Infrastructure (CII), Significant Information Infrastructure (SII). Agencies may customise this template to create their own system-specific System Security Plan or use it as a default System Security Plan for generic low-risk cloud systems.

[Download as Excel](#)

- ✓ System Information
- 2** Controls Configuration
 - Level 0
 - Level 1**
 - Level 2
- 3 Additional Controls
- 4 Summary

Failure to use parameterised interfaces increases the vulnerability to SQL injection or command injection attacks, which can result in the disclosure of sensitive information, the risk of unauthorised access, data manipulation, or even potential system compromise.

Control Configuration

Comply AS-3: Output Sanitisation

Lack of sanitisation for application outputs used in rendering HTML documents exposes the system to the risk of cross-site scripting (XSS) attacks, allowing malicious code execution in users' browsers.

Control Configuration

Control Statement

Sanitise all application outputs that will be used to render a HTML document.

Course of Action

Comply

Comments (Optional)

Empty text input field for comments.

Control Recommendations

We built an SSP editing and submission portal to facilitate central visibility and approvals.

Implementation of Modern Policy Development

DevOps, Publishing and Deployment



Challenges with traditional policy development



Closed source Authoring

Word docs and
Spreadsheets



Internal only

Restricted content not
open to industry



Slow releases

Yearly cycle with multiple manual
approval forums

Our journey to OSCAL



OSCAL Experiments

Standardisation and automation

2023



Cloud-First Architecture

Markdown, Git and Backstage

2022



Launch

Controls open to the industry

2024

Engineering mindset - Adopting open source principles



Transparency

Developed in the open



Interoperability

Machine-readable standards



Community

Engaging internally and with the industry

Product mindset - Policy-as-product



User-centric

Understand stakeholders
and their needs



Data-driven

Improve based on data
collection and analytics



Collaborative

Crowdsourcing inputs and
improvements

Applying Git and DevOps to policy development

Planning

Issue and milestone tracking

Git history, and more importantly, git blame ;)

VS

Multiple word documents emailed back and forth

Contributing

Crowdsourced commits on branches

Merge request reviews by CODEOWNERS

VS

More word documents and large approval committees

Release

Version tags for releases

Automatically generated change logs

VS

Email blasts and manual intranet website updates

Trestle: DevOps tool for maintaining OSCAL projects

```
generate_markdown:
    trestle author catalog-generate --name im8-lite --

assemble_json:
    trestle author catalog-assemble --markdown markdown

add_ssp_control:
    trestle create -f system-security-plans/low-risk-c

merge_catalog:
    cd ./catalogs/im8-lite && \
    trestle merge -e "catalog.*" && \
    cd ../../

split_catalog:
    trestle split -f ./catalogs/im8-lite/catalog.json
    rm -r catalogs/low-risk-resolved-catalog
    rm -r catalogs/medium-risk-resolved-catalog

generate_new_uuid:
    trestle create -f system-security-plans/low-risk-c

assemble_dist: merge_catalog
    trestle assemble catalog -n im8-lite
    trestle assemble profile -n sandbox-level-0
    trestle assemble profile -n sandbox-level-1
    trestle assemble profile -n sandbox-level-2
    trestle assemble profile -n low-risk-level-0
    trestle assemble profile -n low-risk-level-1
    trestle assemble profile -n low-risk-level-2
```



- Splits, merges, and resolves OSCAL models
- Generates final distribution files
- Validates correctness of OSCAL schema

Automatically-generated review website with Next.js

The screenshot shows a web application titled "IMB-reform Portal" with a search bar and a navigation menu. The main content area displays the following sections:

- AS-1: Input Validation**
 - Group: [Application Security](#)
- Control Statement**

Validate all application inputs to ensure that they match the expected type, structure, or format.
- Control Recommendations**

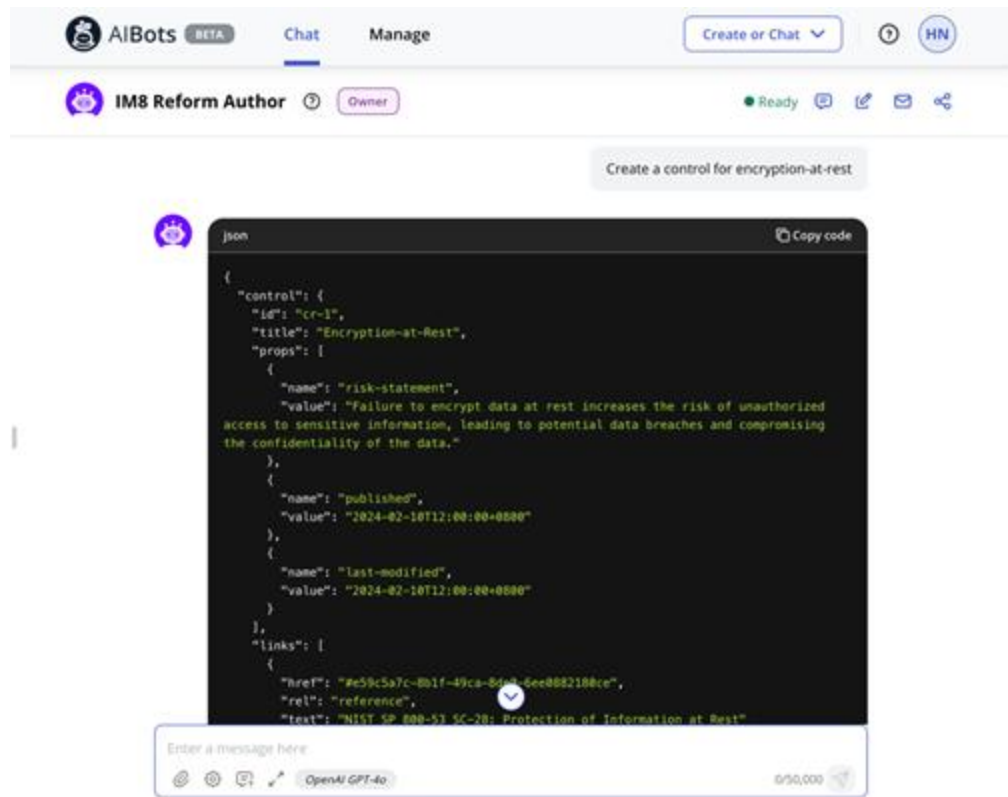
Strictly validating inputs against a comprehensive schema prevents injection attacks caused by inserting special characters or content that would cause the application to perform incorrect operations.
- Risk Statement**

Without input validation, there's a heightened risk of injection attacks, data manipulation, or system crashes due to unexpected input, potentially leading to unauthorised access or disruption of services.
- References**
 - [MvSP 2.5: Security libraries](#)

On the right side, there is a sidebar titled "On This Page" with links to "Control Statement", "Control Recommendations", "Risk Statement", and "References". Below this, there are links for "Question? Give us feedback" and "Edit this page".

- User-friendly view generated from OSCAL JSON files
- Faster search, linking, and grouping
- Built using continuous deployment with every merge to main branch

Assisting policy developers with AI tools

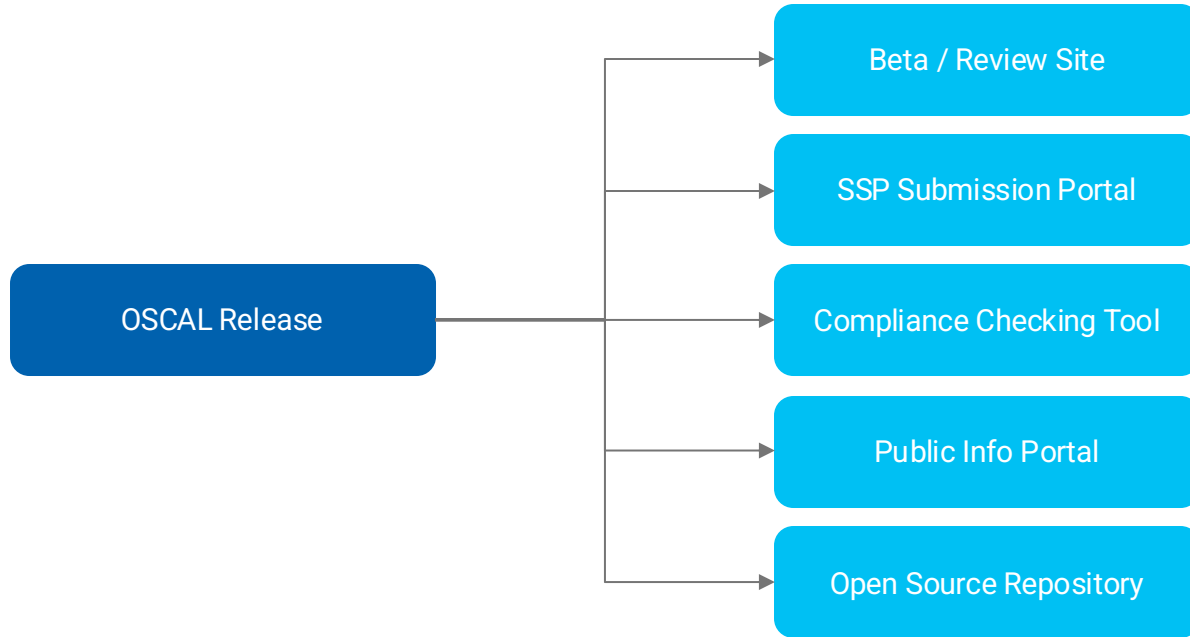


The screenshot shows a chat interface for 'AI Bots BETA'. The user is 'IM8 Reform Author' (Owner). A prompt 'Create a control for encryption-at-rest' is entered. The AI bot, 'OpenAI GPT-4o', outputs the following JSON:

```
json
{
  "control": {
    "id": "cr-1",
    "title": "Encryption-at-Rest",
    "props": [
      {
        "name": "risk-statement",
        "value": "Failure to encrypt data at rest increases the risk of unauthorized access to sensitive information, leading to potential data breaches and compromising the confidentiality of the data."
      },
      {
        "name": "published",
        "value": "2024-02-10T12:00:00+0800"
      },
      {
        "name": "last-modified",
        "value": "2024-02-10T12:00:00+0800"
      }
    ],
    "links": [
      {
        "href": "#59c5a7c-8b1f-49ca-8d90-6ee885218bce",
        "rel": "reference",
        "text": "NIST SP 800-53 SC-28: Protection of Information at Rest"
      }
    ]
  }
}
```

- A custom AI bot outputting controls in OSCAL JSON format.
- Useful starting point for non-technical policy developers.

Publishing to downstream projects from a single source



Lessons learned

- Governance teams are not familiar with Git or JSON. Willing to learn but training and helper tools needed.
- Users approach controls with different goals - some just want to be told what to do, others want greater flexibility because they know what they want.
- LLMs can help accelerate but still need oversight.

Improving the rulebook was only the start...

Where to next?





The road ahead



Continuous Compliance



Compliant  Secure

For Discussion - Modernising audit and compliance

- Beginning to adopt the OSCAL Assessment Layer
- Exploring a framework or SDK for evidence fetchers and checkers with archival storage.
- Evaluating GRC vs Opensource vs Custom built
- Operationalising approval workflows and coupling to profile levels

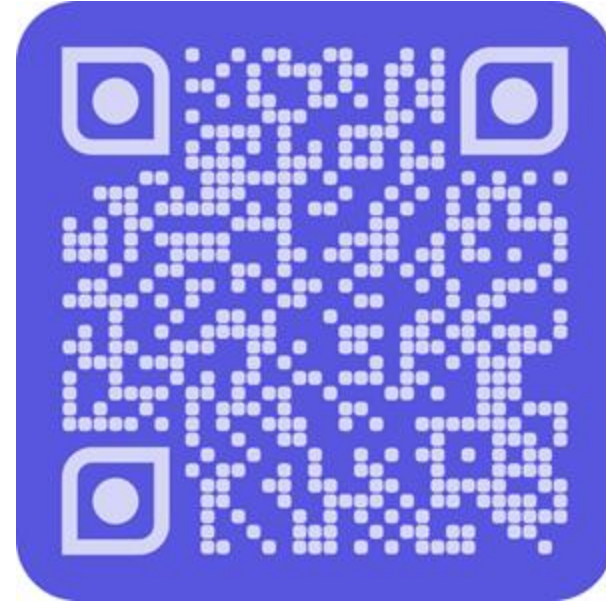
For Discussion - Platforms and shared responsibility

- Govtech runs a number of central platforms. Currently each system running on a platform needs to be assessed from scratch.
- Exploring Components, Certifications and leveraged authorisation.
- Culture and practice change needed for platform teams to share components or SSPs.
- Partner engagement for Component Definitions

Open to the public



Github Repo



Information Portal

GOVTECH
SINGAPORE

