Modernizing SCAP

Gabriel Alford
Member of the Technical Staff
Office of the Chief Technologist
Red Hat Public Sector

The Dark Ages.

Back in the day....

- SRR scripts
- Gold disk
- Multiple other implementations

The Dark Ages.

A Source of Light.

SCAP to the rescue!

- Standardized way of scanning systems
- In 2011, RHEL6 brought collaboration with the scap-security-guide project
    - NSA
    - DISA
    - Integrators
- scap-security-guide project expressed SCAP content in XML Shorthand
- Red Hat & NSA traveled all over the US leading various SCAP workshops

The Dark Ages.

A Source of Light.

Storm on the Horizon.

Why is this XML and why is it so complicated?

- Feedback from workshops
- Feedback from operational sites
- Feedback from users
- Community shrinking

The Dark Ages.

A Source of Light.

Storm on the horizon.

Views of Today.

"SCAP is a complicated and legacy language! Use ours instead."

Feedback from various cloud scanner vendors including those with SCAP-validated scanners

- Tenable audit.rules
- Red Hat Ansible
- Chef Inspec
- Qualys
- Aquasec

The Dark Ages.

A Source of Light.

Storm on the horizon.

Views of Today.

Potential Iteration to 2.0.

- *Make people happy again*

- YAML for authoring
- JSON for final “machine” language format
- Start with XCCDF, OCIL, and OVAL
- Remaining components to follow later

XCCDF 2.0 YAML Rule Example

```yaml
- id: selinux_state

  title: 'Ensure SELinux State is Enforcing'

  description: |-
      The SELinux state should be set to SELINUX={{ var_selinux_state }} at
      system boot time.

  rationale: |-
      Setting the SELinux state to enforcing ensures SELinux is able to confine
      potentially compromised processes to the security policy, which is designed to
      prevent them from causing damage to the system or further elevating their
      privileges.

  severity: high

  identifiers:
      cce: 27334-2

  references:
      disa: 2165,2696
      nist: AC-3,AC-3(3),AC-3(4),AC-4,AC-6,AU-9,SI-6(a)
      srg: SRG-OS-000445-GPOS-00199
```

OCIL 3.0 YAML Example

```
- id: selinux_state

  title: 'Ensure SELinux State is Enforcing'

  audit_question: |-
       Check the file /etc/selinux/config and ensure the following line appears:
       SELINUX=Enforcing

       Is it the case that SELINUX is not set to enforcing?

  audit_action:
       when_true: pass
       when_false: fail
```

OVAL 6.0 YAML Example

```
- id: selinux_state

  title: 'Ensure SELinux State is Enforcing'

  description: |-
      The SELinux state should be set to SELINUX={{ var_selinux_state }} at
      system boot time.

  metadata:
       type: compliance
       platforms:
           - RedHat >= 7
       version: 1

  checks:
    - textfile:
        path: /etc/sysconfig/selinux
        type: file
        pattern: '^[\s]*SELINUX[\s]*=[\s]*(.*)[\s]*$'
        line: 'SELINUX={{ var_selinux_state }}'
        instance: only_one_exists
        state: exists
```

```
- id: selinux_state

  title: 'Ensure SELinux State is Enforcing'

  description: |-
         The SELinux state should be set to SELINUX={{ var_selinux_state }} at
         system boot time.

  rationale: |-
         Setting the SELinux state to enforcing ensures SELinux is able to confine
         potentially compromised processes to the security policy, which is designed to
         prevent them from causing damage to the system or further elevating their
         privileges.

  severity: high

  audit_question: |-
       Check the file /etc/selinux/config and ensure the following line appears:
       SELINUX=Enforcing

       Is it the case that SELINUX is not set to enforcing?

  audit_action:
       when_true: pass
       when_false: fail

  metadata:
       type: compliance
       platforms: RedHat >= 7
       version: 1

  checks:
    - textfile:
```

```
....
        path: /etc/sysconfig/selinux
        type: file
        pattern: '^[\s]*SELINUX[\s]*=[\s]*(.*)[\s]*$'
        line: 'SELINUX={{ var_selinux_state }}'
        instance: only_one_exists
        state: exists

  remediations:
    - bash: |-
        grep -q ^SELINUX= /etc/selinux/config && \
        sed -i 's/SELINUX=.*/SELINUX={{ var_selinux_state }}/g' /etc/selinux/config
        if ! [ $? -eq 0 ]; then
              echo 'SELINUX={{ var_selinux_state }}' >> /etc/selinux/config
        fi

    - puppet: |-
        file_line { 'Ensure SELinux is enabled':
        path  => '/etc/selinux/config',
        line  => 'SELINUX={{ var_selinux_state }}',
        match => '^SELINUX=\w+',
        }

    - chef: |-
        ruby_block 'replace_line' do
          block do
            file = Chef::Util::FileEdit.new('/etc/sysconfig/selinux')
            file.search_file_replace_line('SELINUX=, 'SELINUX={{ var_selinux_state }}')
            file.write_file
          end
        End
```

What this allows

- Easier to understand and edit
- Faster development - tools and content
- Follows industry trends
- An inherent API in the SCAP standard
- Broader database support
- Greater flexibility
  - Tailoring
  - Building SCAP content
  - Smaller file sizes
- Integration opportunities
  - Other standards such as STIX, TAXII, etc.
  - Easily send data to AI and ML frameworks such as Tensorflow

Initial Steps

- Create a top-level GitHub development organization containing **all** the components of the SCAP standard
  - https://github.com/SCAP/xccdf
  - https://github.com/SCAP/oval
  - https://github.com/SCAP/ocil

- SCAP documentation should be written in Markdown or reStructuredText
  - No more monolithic word docs or PDFs for specs as inputs
  - Specification can be generated easily in multiple different output formats

- Update SCAP components to be YAML-based authoring and JSON-based machine language

Final Thoughts

- Don’t reinvent the wheel! Give it new tread!

Questions?