Retired Draft

Warning Notice

The attached draft document has been RETIRED. NIST has discontinued additional development of this document, which is provided here in its entirety for historical purposes.

Retired Date January 21, 2025

Original Release Date May 1, 2015

Retired Document

Status	Initial Public Draft (IPD)
Series/Number	NIST Internal Report 8058
Title	Security Content Automation Protocol (SCAP) Version 1.2 Content Style Guide: Best Practices for Creating and Maintaining SCAP 1.2 Content
Publication Date	May 2015
Additional Information	See https://csrc.nist.gov/projects/security-content-automation-protocol for information on SCAP.

NIST NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
U.S. DEPARTMENT OF COMMERCE

NISTIR 8058 (Draft)

Security Content Automation Protocol (SCAP) Version 1.2 Content Style Guide (Draft)

Best Practices for Creating and Maintaining SCAP 1.2 Content

Harold Booth
Melanie Cook
Stephen Quinn
David Waltermire
Karen Scarfone

National Institute of Standards and Technology
U.S. Department of Commerce

NISTIR 8058 (Draft)

Security Content Automation Protocol (SCAP) Version 1.2 Content Style Guide (Draft)

Best Practices for Creating and Maintaining SCAP 1.2 Content

Harold Booth
Melanie Cook
Stephen Quinn
David Waltermire
Computer Security Division
Information Technology Laboratory

Karen Scarfone
Scarfone Cybersecurity
Clifton, Virginia

May 2015

U.S. Department of Commerce
Penny Pritzker, Secretary

National Institute of Standards and Technology
Willie May, Acting Under Secretary of Commerce for Standards and Technology and Acting Director

National Institute of Standards and Technology Internal Report 8058
42 pages (May 2015)

Certain commercial entities, equipment, or materials may be identified in this document in order to describe an experimental procedure or concept adequately. Such identification is not intended to imply recommendation or endorsement by NIST, nor is it intended to imply that the entities, materials, or equipment are necessarily the best available for the purpose.

There may be references in this publication to other publications currently under development by NIST in accordance with its assigned statutory responsibilities. The information in this publication, including concepts and methodologies, may be used by Federal agencies even before the completion of such companion publications. Thus, until each publication is completed, current requirements, guidelines, and procedures, where they exist, remain operative. For planning and transition purposes, Federal agencies may wish to closely follow the development of these new publications by NIST.

Organizations are encouraged to review all draft publications during public comment periods and provide feedback to NIST. All NIST Computer Security Division publications, other than the ones noted above, are available at http://csrc.nist.gov/publications.

Public comment period: *May 1, 2015* through *June 1, 2015*

National Institute of Standards and Technology
Attn: Computer Security Division, Information Technology Laboratory
100 Bureau Drive (Mail Stop 8930) Gaithersburg, MD 20899-8930
Email: NISTIR8058-comments@nist.gov

Reports on Computer Systems Technology

The Information Technology Laboratory (ITL) at the National Institute of Standards and Technology (NIST) promotes the U.S. economy and public welfare by providing technical leadership for the Nation's measurement and standards infrastructure. ITL develops tests, test methods, reference data, proof of concept implementations, and technical analyses to advance the development and productive use of information technology. ITL's responsibilities include the development of management, administrative, technical, and physical standards and guidelines for the cost-effective security and privacy of other than national security-related information in Federal information systems.

Abstract

The Security Content Automation Protocol (SCAP) is a suite of specifications that standardize the format and nomenclature by which software flaw and security configuration information is communicated, both to machines and humans. SCAP version 1.2 requirements are defined in NIST Special Publication 800-126 Revision 2. Over time, certain stylistic conventions regarding the authoring of SCAP 1.2 content have become best practices. While these best practices are not required, they improve the quality of SCAP content in several ways, such as improving the accuracy and consistency of results, avoiding performance problems, reducing user effort, lowering content maintenance burdens, and enabling content reuse. This document has been created to capture the best practices and encourage their use by SCAP content authors and maintainers.

Keywords

information security; SCAP content; SCAP data stream; SCAP programmer; SCAP style guide; security automation; Security Content Automation Protocol (SCAP)

Acknowledgements

The authors wish to thank their colleagues who reviewed drafts of this document and contributed to its technical content. In addition to the authors, other sources of best practices included presentations from Kent Landfield of McAfee [9] and Shane Shaffer of G2, Inc. [10], and several code examples were derived from the United States Government Configuration Baseline (USGCB) checklist for Microsoft Windows 7 [11].

Trademark Information

OVAL and CVE are registered trademarks, and CCE, CPE, and OCIL are trademarks, of The MITRE Corporation.

All other registered trademarks or trademarks belong to their respective organizations.

Table of Contents

1 Introduction 1
1.1 Purpose and Scope 1
1.2 Audience 1
1.3 Document Structure 1
1.4 Document Conventions 1
2 Overview of SCAP 1.2 Stylistic Concepts 3
3 Best Practice Template 4
4 General Style Best Practices 5
4.1 When writing content, use the latest version of the SCAP specification. 5
4.2 Test all content. 5
4.3 Run the SCAP Content Validation Tool on all content and remove warnings whenever feasible. 6
4.4 Avoid unnecessarily including dynamic information in content. 6
4.5 Use specific properties instead of overloading general properties. 7
4.6 Spell check all text that might be presented to the user. 8
4.7 When reusing content, recognize its originator. 9
4.8 Explicitly specify all default attributes when creating content that will be signed. 9
5 OCIL Style Best Practices 10
5.1 Only include one fact per question. 10
5.2 Sequence questions to avoid asking unnecessary questions. 12
5.3 Provide step-by-step instructions when helpful. 12
5.4 Use <ocil:choice_question> instead of <ocil:string_question> when feasible. 13
5.5 Use *<ocil:choice_group>* when feasible. 14
6 OVAL Style Best Practices 16
6.1 Check for the conditional applicability of vulnerabilities. 16
6.2 Include concise comments in elements whenever possible. 17
6.3 Use safe regular expressions in pattern matching. 18
6.4 Consider performance impacts when writing or modifying checks. 18
6.5 When feasible, write one check that applies to multiple software versions, instead of duplicate checks for each version. 19
6.6 Use external variables so a single check can be used for multiple input

variables. 19
6.7 When creating an external variable, carefully consider the possible values. 20
6.8 Reuse check content where possible. 21
6.9 Indicate revisions of definitions, tests, objects, states, and variables. 22
6.10 Have a single CCE or CVE per definition when applicable. 23
6.11 Be careful when extending extended definitions. 24
6.12 Explicitly declare the `<oval:registry_state>` element's `<oval:type>` element. 24
6.13 Avoid the use of deprecated tests. 25
6.14 Ensure that the schema location and version number agree. 25
7 XCCDF Style Best Practices 27
7.1 Use a tailoring document when deriving your own XCCDF content from someone else's benchmark. 27
7.2 Indicate revisions of a single benchmark or tailoring document. 28
7.3 Indicate revisions of <xccdf:Profile>, <xccdf:Group>, <xccdf:Rule>, and <xccdf:Value> elements. 29
7.4 When referencing OVAL from XCCDF, match datatypes. 30
7.5 Have a single CCE or CVE per rule when applicable. 30
7.6 If a patch checklist is required, use separate checklists for patches and configuration settings. 31
8 SCAP Data Stream Style Best Practices 32
8.1 Avoid using data stream identifiers to convey other information to automated parsers. 32
9 Best Practice Topics for Further Discussion 33
9.1 Is it preferable to use plaintext or XHTML? 33

List of Appendices

Appendix A— Acronyms and Abbreviations 34
Appendix B— References 35

1 Introduction

1.1 Purpose and Scope

The purpose of the document is to provide a list of best practices for Security Content Automation Protocol (SCAP) version 1.2 content developers and maintainers. NIST encourages the adoption of these best practices. These best practices are not SCAP requirements (which are defined in NIST Special Publication (SP) 800-126 Revision 2 [1]), but rather they are recommendations that help ensure greater SCAP content reuse and interoperability with SCAP consumers.

1.2 Audience

The intended audience for this document is individuals who have responsibilities for creating, maintaining or verifying SCAP 1.2 content. This includes technical subject matter experts, programmers, SCAP content authors, and SCAP content consumers. It is assumed that readers are already familiar with NIST SP 800-126 Revision 2 [1].

1.3 Document Structure

The remainder of this document is organized into the following major sections and appendices:

- Section 2 elaborates on the need for an SCAP content style guide to supplement NIST SP 800-126, which specifies requirements for SCAP version 1.2 content.
- Section 3 defines the fields of the template used for discussing best practices throughout the rest of the document.
- Section 4 provides details on best practices that apply to all the SCAP languages: Extensible Configuration Checklist Description Format (XCCDF), Open Vulnerability and Assessment Language (OVAL), and Open Checklist Interactive Language (OCIL).
- Section 5 focuses on best practices for OCIL.
- Section 6 covers best practices for OVAL.
- Section 7 addresses best practices for XCCDF.
- Section 8 discusses best practices for SCAP data streams.
- Section 9 details best practice topics that need community discussion before further development.
- Appendix A lists acronyms and abbreviations used throughout the document.
- Appendix B provides the references for the document.

1.4 Document Conventions

Some of the requirements and conventions used in this document reference Extensible Markup Language (XML) content [6]. An example of a reference is: Explicitly declare the `<oval:registry_state>` element's `<oval:type>` element. In this example the notation `<oval:registry_state>` can be replaced by the more verbose equivalent "the XML element whose qualified name is `oval:registry_state`".

The general convention used when describing XML attributes within this document is to reference the attribute as well as its associated element including the namespace alias, employing the general form "*@attributeName* for the *<prefix:localName>*".

See Table 1 of NIST SP 800-126 Revision 2 [1] for the conventional XML mappings used for SCAP 1.2 content.

2 Overview of SCAP 1.2 Stylistic Concepts

SCAP 1.2 includes several expression language component specifications: XCCDF [2], OVAL [4], and OCIL [3]. Each of these specifications includes robust feature sets that ensure broad application and flexibility for their individual use cases. To ensure greater interoperability for SCAP content authors and consumers, particularly when using multiple component specifications together, the SCAP specification (documented in NIST SP 800-126 Revision 2 [1]) adds constraints to the component specifications in the form of SCAP 1.2 requirements. For example, XCCDF is a flexible XML specification, but this flexibility needed to be constrained through additional SCAP requirements to ensure that SCAP-validated products could process XCCDF for a particular set of use cases.

An example of such a constraint is from Section 3.2.2 of NIST SP 800-126 Revision 2: "The `<xccdf:version>` element and the `@id` attribute SHALL be used together to uniquely identify all revisions of a benchmark." While the use of the `<xccdf:version>` element and the `@id` attribute are both required by the XCCDF specification, the requirement to use them together to uniquely identify benchmark revisions is not part of the XCCDF specification. It has been added through NIST SP 800-126 Revision 2 as an SCAP-specific requirement.

Over time, certain stylistic conventions regarding the authoring of SCAP content have become informal best practices. An example is using a tailoring document when deriving your own XCCDF content from someone else's benchmark. While these best practices are not required by NIST SP 800-126 Revision 2 or any of the component specifications, the best practices improve the quality of SCAP content in several ways, such as:

- Improving the accuracy and consistency of results
- Avoiding performance problems
- Reducing user effort
- Lowering content maintenance burdens
- Enabling content reuse

This document has been created to capture the best practices and encourage their use by SCAP content authors and maintainers.

Nothing in this document contradicts the requirements of NIST SP 800-126 Revision 2 and the component specifications.

3 Best Practice Template

Sections 4 through 9 of this document follow the template defined in this section for discussing each best practice. The possible fields are listed in order below. Note that this template may be used by readers to submit their own best practice suggestions to NIST for possible inclusion in revisions of this document.

x.x This is a best practice statement. Mandatory. The best practice statement expresses the best practice in a concise sentence.

Rationale: Mandatory. This states in a sentence the reason why the best practice is being recommended.

Background: Optional. This gives the reader background information necessary to understand the rest of the discussion, such as indicating which elements being discussed are mandatory and which are optional according to the SCAP specification or component specifications.

Reference: Optional. This points the reader to additional sources of information on the topic.

Dependencies: Optional. This lists any dependencies that this best practice has on other best practices.

Applicability: Mandatory. This speaks to the situations for which this best practice is recommended.

Implementation: Mandatory. This explains how the reader can best go about performing this best practice.

Impact/Consequence: Mandatory. This describes the impact of following the best practice and/or the consequence of not following the best practice.

Example: Optional. This contains excerpts of SCAP content to better illustrate the best practice through an example. Some content is omitted for brevity; omissions are indicated through "…" notation.

4 General Style Best Practices

This section discusses general style best practices (those that apply to XCCDF, OVAL, and OCIL).

4.1 When writing content, use the latest version of the SCAP specification.

Rationale: Using the latest version of SCAP and its component specifications supports greater interoperability and functionality.

Reference: As of this writing, the latest version of SCAP is 1.2, which is defined in NIST SP 800-126 Revision 2 [1]. The versions of the component specifications, such as OCIL, OVAL, and XCCDF, are defined in Section 2 of NIST SP 800-126 Revision 2.

Applicability: This applies to any situation where new content is being developed. This best practice is not meant to imply that all existing content should be updated to the latest SCAP version, although in many cases doing so will take little effort.

Implementation: Develop all new content using the latest SCAP version and the associated versions of its component specifications. An example is OVAL. SCAP 1.2 specifies the use of OVAL 5.10. Although older versions of OVAL content may be used, new OVAL content should be developed in OVAL 5.10, not deprecated OVAL versions.[1]

Impact/Consequence: This best practice supports interoperability by recommending the use of the latest SCAP specification and its associated component specifications instead of older specifications. Older specifications are likely to lose support much earlier than newer specifications. Also, newer specifications tend to have greater functionality, allowing content to be written more effectively and efficiently than with previous specifications.

4.2 Test all content.

Rationale: Testing all SCAP content reduces the number of errors in final content, thus improving the performance, consistency, and accuracy of the content.

Applicability: This applies to any situation where new content is being developed or existing content is being modified.

Implementation: It is important to ensure that content you develop or customize works correctly in all possible cases, to the extent that this is feasible. This requires testing the content.

Impact/Consequence: Obviously content that doesn't work at all or doesn't work properly can cause a variety of negative impacts, such as unreliable or incorrect results, or performance

[1] Since the release of the SCAP 1.2 specification [1], OVAL 5.10 was updated to OVAL 5.10.1 for bug fixes. References within this document to OVAL 5.10 are intended to imply the use of OVAL 5.10.1

problems. By performing thorough testing of content, users of that content can be spared a variety of problems.

4.3 Run the SCAP Content Validation Tool on all content and remove warnings whenever feasible.

Rationale: Correcting content that is generating validation warnings improves the interoperability of content.

Background: From the SCAP Specifications page (http://scap.nist.gov/revision/1.2/): "The SCAP Content Validation Tool is designed to validate the correctness of a SCAP data stream for a particular use case according to what is defined in SP 800-126."

Reference: For more information on the SCAP Content Validation Tool (SCAPval), see the Tools section of http://scap.nist.gov/revision/1.2/.

Applicability: This is applicable to all SCAP content that is written or edited.

Implementation: Run the SCAP Content Validation Tool on all new or revised content and review the warnings for the content. For all feasible warnings, modify the content so that the warnings will no longer be generated. Note that it may not be possible to eliminate all warnings in SCAP content. An example is referencing a Common Platform Enumeration (CPE) entry that is not contained in the official CPE dictionary.

Impact/Consequence: This best practice supports interoperability by ensuring that SCAP content is as consistent with the specifications and general expectations of SCAP style as feasible. If warnings are not removed from content, this could cause unpredictable behavior in certain tools that are not expecting these associated conditions to occur.

4.4 Avoid unnecessarily including dynamic information in content.

Rationale: Examples of dynamic information are vulnerability scores and security control mappings and text. Dynamic information should be linked to through associated identifiers instead of embedding it within the SCAP content because of the maintenance burden.

Reference: See Section 3.2.4.4 of NIST SP 800-126 Revision 2 [1] for more information on mapping to vulnerability scores, and Section 3.6 for information on security control mappings.

Applicability: This applies whenever dynamic information might be inserted into content, not just for vulnerability scores and security control mappings.

Implementation: NIST SP 800-126 Revision 2 provides insights into how this would be implemented for vulnerability scores and security control mappings and text. From Section 3.6 regarding security control text: "A preferred technique is to embed only the CCE identifiers within SCAP content; when mappings to NIST SP 800-53 control identifiers are needed, dynamically acquire them from the official data feed and associate them to the SCAP content based on its embedded CCE identifiers." From Section 3.2.4.4 regarding vulnerability scores:

"During scoring, current CVSS scores acquired dynamically, such as from a data feed, SHOULD be used in place of the `@weight` attribute within XCCDF vulnerability-related rules." The same principle applies to any other forms of dynamic content.

Impact/Consequence: Embedding dynamic information in content causes a significant maintenance burden. This is particularly true with vulnerability scores, which may change over time, but it is also relevant for security control mappings and text, such as from NIST SP 800-53. Although NIST SP 800-53 does not change frequently, it has many pages of content that would unnecessarily need to be duplicated in SCAP content if mappings through identifiers were not used. Duplicating this content increases the chance of errors, takes considerable time, and necessitates editing the content whenever a new version of NIST SP 800-53 or related errata is released.

Example: The *<xccdf:ident>* element in the abbreviated XCCDF example below shows the use of a CCE identifier instead of hard-coded CCE information. The CCE identifier can be used to dynamically look up the current CCE information.

```
<xccdf:Rule id="xccdf_gov.nist_rule_account_lockout_duration"
selected="false">
  <xccdf:title>...</xccdf:title>
  <xccdf:description>...</xccdf:description>
  <xccdf:reference>...</xccdf:reference>
  <xccdf:ident system="http://cce.mitre.org">CCE-9308-8</xccdf:ident>
  <xccdf:check system="http://oval.mitre.org/XMLSchema/oval-
definitions-5">...</xccdf:check>
</xccdf:Rule>
```

Another example shows how a CCE identifier can be referenced from within OCIL content by using an *<ocil:reference>* element.

```
<questionnaire id="ocil:usgcb.win7.checklist:questionnaire:1">
  <title>USGCB Windows 7 User Settings: Question 1</title>
  <description>Enable screen saver</description>
  <references>
    <reference href="http://cce.mitre.org">CCE-10051-1</reference>
  </references>
  <actions>
    <test_action_ref>ocil:usgcb.win7.checklist:testaction:1
</test_action_ref>
  </actions>
</questionnaire>
```

4.5 Use specific properties instead of overloading general properties.

Rationale: Overloading a property instead of using an existing property makes the information stored within it less readily accessible.

Applicability: This applies whenever a specific property exists that is well suited for the information that the content author wants to store.

Implementation: When there is a more specific property and a more general property available that information could be stored in, use the more specific property. An example is the *<xccdf:Group>* element. This element has a general *<xccdf:description>* property, which is defined in the XCCDF specification [2] as "text that describes the item." The *<xccdf:Group>* element also has several more specific properties, such as *<xccdf:warning>*, which is "a note or caveat about the item intended to convey important caution information for the benchmark user;" and *<xccdf:rationale>*, which is "descriptive text giving rationale or motivations for abiding by this group/rule." A warning should be stored in the *<xccdf:warning>* element, not the *<xccdf:description>* element.

Impact/Consequence: Using specific properties instead of more general properties makes it easier for both tools and humans to find the information of interest to them.

Example:

```
<Group id="xccdf_gov.sample_group_filepermissions">
  <description>This group contains rules pertaining to file
permissions</description>
  <warning>File permission settings contained within the following
rules may cause application errors</warning>
  <rationale>Maintaining proper file permissions is critical
to...</rationale>
  ...
</Group>
```

4.6 Spell check all text that might be presented to the user.

Rationale: Spell checking text visible to the user promotes readability and understanding of the text.

Applicability: This applies in all cases where text might be presented to a user, including comments.

Implementation: It is important to check the text of all elements presenting text to the user for any misspellings, typos, etc. This can be accomplished by loading the content into a tool that has spell checking capabilities. However, authors are advised to manually proofread their text as well to catch other errors that cannot be caught through spell checking.

Impact/Consequence: This helps ensure that text is clear, so that the users will understand them. Ensuring that text is spelled correctly also creates a professional impression and helps to underscore the seriousness and legitimacy of the materials.

4.7 When reusing content, recognize its originator.

Rationale: The original author of content should be recognized for their efforts.

Applicability: This applies whenever reusing content from another party.

Implementation: SCAP component specifications do not have a specific property for recognizing the originator of content, but the various specifications have comment attributes (e.g., OVAL), metadata attributes (e.g., XCCDF), or other text field attributes that could be used to give credit to the source of the content.

Impact/Consequence: Recognizing the originator of the content is the ethical thing to do. It may also be required because of the content's licensing model. Failure to recognize the originator could cause ethical questions to be raised and could be a violation of the content license.

4.8 Explicitly specify all default attributes when creating content that will be signed.

Rationale: Some parsers automatically fill in the values of default attributes before signing content, so if default attributes are not provided, signature verification will fail for other parsers that do not automatically fill in the values.

Applicability: This best practice applies whenever digitally signing an SCAP data stream or other SCAP content.

Implementation: Explicitly provide values for all default attributes instead of assuming the default values.

Impact/Consequence: If all default attributes are not explicitly defined when digitally signing SCAP content, certain parsers may fail to process the data stream signing correctly. This could lead to processing errors or a failure to recognize the legitimacy of signed content.

5 OCIL Style Best Practices

This section discusses style best practices specific to OCIL.

5.1 Only include one fact per question.

Rationale: Having a single fact per question means that the answer to the question will provide a granular answer for a specific fact, not a general answer for a group of facts.

Applicability: This applies in all cases where questions are being written.

Implementation: It may be prudent to break a single question[2] into multiple questions. For example, you might want to ask a user whether the system's password policy for service accounts mandates that passwords are at least 15 characters long and meet complexity requirements. This should be broken into at least two questions: 1) does the system mandate that passwords for service accounts are at least 15 characters long?, and 2) does the system mandate that passwords for service accounts meet complexity requirements? It may be necessary to break the complexity requirements question into multiple questions, depending on the nature of those requirements. You may also want to first ask if the system enforces a password policy, so as to skip all other password policy-related questions if it does not.

Impact/Consequence: By having a single fact per question, the information provided by answering the questions is much more granular and actionable (for example, an answer indicating that the system does mandate a minimum password length of 15 characters, but does not mandate password complexity requirements, instead of an answering simply indicating that the system does not meet the password policy.) Questions are also clearer for the user to answer because only a single fact is being considered at any given time, so users are more likely to provide accurate answers.

[2] The *<ocil:question>* element is abstract and does not appear in OCIL content. Instead, a question is represented as one of the following four elements: *<ocil:boolean_question>*, *<ocil:choice_question>*, *<ocil:numeric_question>*, or *<ocil:string_question>*.

Example: The code below shows how multiple *<ocil:boolean_question>* elements can be used to achieve more granular results.

```
<questionnaires>
  <questionnaire id="ocil:namespace_here:questionnaire:1">
    <title>Insurance policy coverage</title>
    <actions>
      <test_action_ref>ocil:namespace_here:testaction:1
      </test_action_ref>
      <test_action_ref>ocil:namespace_here:testaction:2
      </test_action_ref>
    </actions>
  </questionnaire>
</questionnaires>
<test_actions>
  <boolean_question_test_action
    question_ref="ocil:namespace_here:question:1"
    id="ocil:namespace_here:testaction:1">
    <when_true>
      <result>PASS</result>
    </when_true>
    <when_false>
      <result>FAIL</result>
    </when_false>
  </boolean_question_test_action>
  <boolean_question_test_action
    question_ref="ocil:namespace_here:question:2"
    id="ocil:namespace_here:testaction:2">
    <when_true>
      <result>PASS</result>
    </when_true>
    <when_false>
      <result>FAIL</result>
    </when_false>
  </boolean_question_test_action>
</test_actions>
<questions>
  <boolean_question id="ocil:namespace_here:question:1">
    <question_text>Does the insurance policy include coverage for
floods?</question_text>
  </boolean_question>
  <boolean_question id="ocil:namespace_here:question:2">
    <question_text>Does the insurance policy include coverage for
earthquakes?</question_text>
  </boolean_question>
</questions>
```

5.2 Sequence questions to avoid asking unnecessary questions.

Rationale: The answer to one question may negate the need to ask other questions, so it is more efficient for users if questions are properly sequenced so that unneeded questions are not asked.

Applicability: This applies in cases where questions are being written and the answer to one or more questions may negate the need to ask other questions.

Implementation: Link test actions so that they ask questions in a series when there are dependencies between those questions. An example is asking a user about a system's password policy characteristics. It may be prudent to first ask the user if the system has a password policy, and only if that answer is in the affirmative, then asking the user about the details of that password policy.

Impact/Consequence: Sequencing questions in this way eliminates asking unneeded questions, which speeds the answering process for users and reduces user frustration.

5.3 Provide step-by-step instructions when helpful.

Rationale: Step-by-step instructions can aid the reader in answering questions.

Background: NISTIR 7692 [3] states in Section 6.5: "Authors SHOULD use instructions elements for questions that users are likely to answer more accurately and/or easily with step-by-step instructions."

Applicability: This is a best practice to consider when writing questions that necessitate user actions, such as manually verifying a setting on a system.

Implementation: Rather than assuming that a user knows how to manually check a system for a particular setting, for example, provide the user with step-by-step instructions using the `<ocil:instructions>` element on how to perform that manual check.

Impact/Consequence: Step-by-step instructions help ensure that users perform the check correctly and consistently, thus leading to higher accuracy in answers. Providing step-by-step instructions may also reduce user frustration and also reduce the amount of time that users need to answer each question.

Example:

```
<boolean_question id="ocil:namespace_here:question:3">
  <question_text>Is the engine oil level low?</question_text>
  <instructions>
    <title>Instructions</title>
    <step><description>Open the hood of the
vehicle</description></step>
    <step><description>Locate the dipstick</description></step>
    <step><description>Remove the dipstick</description></step>
    <step><description>Wipe all oil off the
dipstick</description></step>
    <step><description>Re-insert the dipstick</description></step>
    <step><description>Remove the dipstick</description></step>
    <step><description>Observe the level of oil relative to the mark
on the dipstick indicating the minimum oil level</description></step>
    <step><description>If below the minimum level, respond "Yes",
otherwise respond "No"</description></step>
  </instructions>
</boolean_question>
```

5.4 Use <ocil:choice_question> instead of <ocil:string_question> when feasible.

Rationale: Forcing users to choose from a list of answers instead of typing in an answer can improve the accuracy of answers and reduce the workload for the users.

Applicability: This applies whenever a question is being written that has a small, predefined set of possible answers.

Implementation: It is recommended to use an *<ocil:choice_question>* element when an *<ocil:string_question>* could be used but would have only a small, predefined set of possible answers. Imagine asking users to manually enter the name of their organizational unit to answer an *<ocil:string_question>*. This is likely to generate all sorts of responses that vary based on spelling errors, punctuation differences, and other variations in how people type in strings. Such variation can prevent accurate correlation of data collected from multiple individuals. It would be highly preferable to instead have an *<ocil:choice_question>* defined that lists the organizational units, so that users can simply pick the correct organizational unit.

Impact/Consequence: This reduces the time that it takes users to enter a response. It also improves the consistency and accuracy of the responses by bounding the choices that users have to pick from, instead of allowing free-form text entry. Logic within the *<ocil:string_question_test_action>* element might have to be quite complex to handle capitalization variations and other differences between free-form text entries. A possible disadvantage of using an *<ocil:choice_question>* is if the list of choices itself needs to change frequently. This could cause a maintenance burden, and the tradeoff between consistent input and question maintenance would have to be considered.

Example: Instead of the following:

```
<string_question id="ocil:namespace_here:question:4">
  <question_text>What is your favorite day of the
week?</question_text>
</string_question>
```

Do this:

```
<choice_question id="ocil:namespace_here:question:4">
  <question_text>What is your favorite day of the
week?</question_text>
  <choice id="ocil:namespace_here:choice:1">Sunday</choice>
  <choice id="ocil:namespace_here:choice:2">Monday</choice>
  <choice id="ocil:namespace_here:choice:3">Tuesday</choice>
  <choice id="ocil:namespace_here:choice:4">Wednesday</choice>
  <choice id="ocil:namespace_here:choice:5">Thursday</choice>
  <choice id="ocil:namespace_here:choice:6">Friday</choice>
  <choice id="ocil:namespace_here:choice:7">Saturday</choice>
</choice_question>
```

5.5 Use *<ocil:choice_group>* when feasible.

Rationale: Defining a set of choices once and reusing that set is more efficient and less error-prone than redefining the same set of choices multiple times.

Background: As defined in NISTIR 7692, Section 5.1, an *<ocil:choice_group>* "represents a reusable set of choices for a choice_question. A choice_question MAY reference a choice_group or explicitly specify allowed choices."

Applicability: This applies in all cases where multiple *<ocil:choice_question>* elements are being written and they share the same set of answers.

Implementation: It is recommended to use *<ocil:choice_group>* when the same set of choices is to be used for multiple questions: for example, Always, Usually, Sometimes, Rarely, Never. By placing these in an *<ocil:choice_group>* element, the *<ocil:choice_question>* elements can simply reference the *<ocil:choice_group>* element, instead of each question having the same choices individually defined.

Impact/Consequence: This reduces the amount of effort for the content author and reduces the risk of having typos or other errors in the duplicate sets of choices by giving the author only a single set to write and proofread. This also simplifies the content itself and makes it easier for maintainers—for example, if the example set of choices listed above needed to change, it could be changed in one spot instead of many spots.

Example:

```
<choice_question id="ocil:namespace_here:question:5">
  <question_text>What is your favorite day of the
week?</question_text>
  <choice_group_ref>ocil:namespace_here:choicegroup:1
  </choice_group_ref>
</choice_question>
<choice_question id="ocil:namespace_here:question:6">
  <question_text>What day of the week were you born?</question_text>
  <choice_group_ref>ocil:namespace_here:choicegroup:1
  </choice_group_ref>
</choice_question>
<choice_group id="ocil:namespace_here:choicegroup:1">
  <choice id="ocil:namespace_here:choice:1">Sunday</choice>
  <choice id="ocil:namespace_here:choice:2">Monday</choice>
  <choice id="ocil:namespace_here:choice:3">Tuesday</choice>
  <choice id="ocil:namespace_here:choice:4">Wednesday</choice>
  <choice id="ocil:namespace_here:choice:5">Thursday</choice>
  <choice id="ocil:namespace_here:choice:6">Friday</choice>
  <choice id="ocil:namespace_here:choice:7">Saturday</choice>
</choice_group>
```

6 OVAL Style Best Practices

This section will discuss style best practices specific to OVAL.

6.1 Check for the conditional applicability of vulnerabilities.

Rationale: It is best to ensure that software is present on a system before checking for vulnerabilities in that software.

Background: In the OVAL Definitions Model (Section 4.3 of the OVAL Language Specification [4]), the CriteriaType, CriterionType, and ExtendDefinitionType include an `<oval:applicability_check>` attribute. An optional attribute, `<oval:applicability_check>` is defined as "a boolean flag that when *'true'* indicates that the [criteria|criterion|ExtendDefinition] is being used to determine whether the OVAL Definition applies to a given system. No additional meaning is assumed when *'false'*."

Applicability: This applies in any case where vulnerability criteria were written under the assumption that the user already knows that the potentially affected software is present.

Implementation: This is best explained through an example. Suppose that there is a vulnerability in Acme Enterprise before version 1234. If you didn't use `<oval:applicability_check>` and you used criteria that checked for a version of Acme before 1234, you'd get a true result if you were running Acme version 1230, and a false result if you were running Acme version 1235. But what result would you get if the system didn't have Acme installed? You wouldn't have any way of differentiating this result from an actual true or false value. To prevent this ambiguity from occurring, it is recommended that you set `<oval:applicability_check>` to true; this will cause the absence of software to generate a Not Applicable result.

Impact/Consequence: Following this practice improves the consistency and accuracy of OVAL results.

Example:

```
<definition class="compliance"
id="oval:gov.nist.usgcb.windowsseven:def:1" version="2">
...
  <criteria operator="AND">
    <extend_definition comment="Windows 7 is installed"
definition_ref="oval:gov.nist.cpe.oval:def:1"
applicability_check="true"/>
    <criteria operator="OR">
      <criterion comment="Account Lockout Duration is set to keep
accounts locked until unlocked by an administrator"
test_ref="oval:gov.nist.usgcb.windowsseven:tst:60070"/>
      <criteria operator="AND">
        <criterion comment="Account Lockout Duration is set to keep
accounts locked for at least the profile defined number of minutes"
test_ref="oval:gov.nist.usgcb.windowsseven:tst:60071"/>
        <criterion comment="Profile does not require administrator
unlock" test_ref="oval:gov.nist.usgcb.windowsseven:tst:60072"/>
      </criteria>
      <criterion comment="Account Lockout Duration is set to keep
accounts locked until unlocked by an administrator"
test_ref="oval:gov.nist.usgcb.windowsseven:tst:60073"/>
    </criteria>
  </criteria>
</definition>
```

6.2 Include concise comments in elements whenever possible.

Rationale: Comments help authors, maintainers, and even users of the content to understand what the content is intended to do and to troubleshoot problems that occur.

Background: In the OVAL Definitions Model (Section 4.3 of the OVAL Language Specification [4]), many types, including the CriteriaType, CriterionType, ExtendDefinitionType, TestType, ObjectType, StateType, and VariableType include an *<oval:comment>* property. Some of these *<oval:comment>* properties are mandatory, while others are optional.

Applicability: This applies to writing or editing a wide variety of OVAL elements.

Implementation: Whenever an *<oval:comment>* property is available for an OVAL element, it should be used to provide concise comments for content authors and maintainers. Comments serve as the documentation for OVAL content.

Impact/Consequence: Comments are beneficial for those individuals who are authoring, maintaining, or troubleshooting the content. By having comments, problems are likely to be resolved more quickly and effectively. Comments are also searchable in the XML source, which can aid in content authoring, maintenance, and troubleshooting. Also, well-commented OVAL content is more likely to be reused because its purpose and function are clearly stated.

Example: The OVAL example below shows comments for both the *<oval:extend_definition>* and *<oval:criterion>* elements.

```
<definition class="compliance"
id="oval:gov.nist.usgcb.windowsseven:def:1" version="2">
  <metadata>...</metadata>
  <criteria operator="AND">
    <extend_definition comment="Windows 7 is installed"
definition_ref="oval:gov.nist.cpe.oval:def:1"/>
    <criteria operator="OR">
      <criterion comment="Account Lockout Duration is set to keep
accounts locked until unlocked by an administrator"
test_ref="oval:gov.nist.usgcb.windowsseven:tst:60070"/>
      ...
    </criteria>
  </criteria>
</definition>
```

6.3 Use safe regular expressions in pattern matching.

Rationale: Using safe regular expressions helps ensure that only legitimate inputs are processed.

Applicability: This applies whenever writing or modifying OVAL content that uses pattern matching.

Implementation: Inputs may contain data that is corrupted, malicious, or otherwise unexpected. To handle such inputs properly when doing pattern matching, it is prudent to use safe regular expressions that ensure that only input that meets the specified requirements is further processed.

Impact/Consequence: If inputs are not checked, unexpected inputs may be processed. This could cause tools to crash or produce unpredictable results. If the unexpected inputs are malicious, they could cause the tool to return false results, such as failing to report the existence of exploitable vulnerabilities that attackers could then target.

Example: The *<oval:value>* element below shows an example of a safe pattern matching expression.

```
<registry_state xmlns="http://oval.mitre.org/XMLSchema/oval-
definitions-5#windows" comment="The registry key matches with Windows
7" id="oval:org.mitre.oval:ste:5027" version="4">
  <value operation="pattern match">^[a-zA-Z0-
9\(\)\s]*[Ww][Ii][Nn][Dd][Oo][Ww][Ss] 7[a-zA-Z0-9\(\)\s]*$</value>
</registry_state>
```

6.4 Consider performance impacts when writing or modifying checks.

Rationale: Running certain checks in production environments may cause denial of service conditions to occur because of excessive resource utilization.

Applicability: This applies whenever writing or modifying a check that does not scale well for larger environments. An example is resolving groups on a local host versus a million-host domain.

Implementation: When writing or modifying checks, consider not just the best case or the typical case, but the worst case. If you suspect that there may be negative performance impacts to users, document these within the check. Where possible, consider alternate approaches to authoring the check to reduce the assessment workload.

Impact/Consequence: Failure to consider performance impacts in a variety of environments could cause denial of service conditions in some production environments that use the checks.

6.5 When feasible, write one check that applies to multiple software versions, instead of duplicate checks for each version.

Rationale: This best practice reduces the number of checks that need to be written.

Applicability: This applies whenever you have an opportunity to use the same check on multiple operating system versions or application versions.

Implementation: Create a single check and use it for multiple operating system versions (e.g., Windows 7 and 8) or multiple application versions instead of creating a separate duplicate check for each operating system or application version.

Impact/Consequence: This allows a single check to be used instead of multiple checks, so it reduces the number of checks that need to be written. This makes content maintenance and troubleshooting easier, and it reduces the likelihood of errors entering the content by eliminating the writing of unnecessary checks.

6.6 Use external variables so a single check can be used for multiple input variables.

Rationale: This best practice reduces the number of checks that need to be written.

Reference: For more information on the definition of an OVAL external variable, see Section 4.3.23 of the OVAL specification [4].

Applicability: This applies whenever you have an opportunity to use multiple input variables with a single check, instead of creating multiple checks.

Implementation: Create a single check with external variables instead of duplicate checks with local variables. An example is checking a password length policy. If the OVAL has the minimum length policy hardcoded and there is not an external variable for it, then every time the policy changes, the OVAL has to be changed. This is particularly problematic if other parties will be reusing the content or if there are multiple policies within a single organization (for example, different length requirements for each system security level).

Impact/Consequence: This allows a single check to be used with multiple input variables, so it reduces the number of checks that need to be written. This makes content maintenance and troubleshooting easier, and it reduces the likelihood of errors entering the content by eliminating the writing of unnecessary checks.

Example: This example shows a declaration of an *<oval:external_variable>* element, then an *<xccdf:refine-value>* that declares a value of "12 characters", and then an *<xccdf:Rule>* element declaration that references the external variable and uses the value.

```
<oval:external_variable comment="Minimum Password Length is greater
than or equal to the prescribed value" datatype="int"
id="oval:gov.nist.usgcb.windowsseven:var:22" version="2"/>
...
<xccdf:refine-value
idref="xccdf_gov.nist_value_password_minimum_length_var"
selector="12_characters"/>
...
<xccdf:Rule id="xccdf_gov.nist_rule_minimum_password_length"
selected="false" weight="10.0">
  ...
  <xccdf:check system="http://oval.mitre.org/XMLSchema/oval-
definitions-5">
    <xccdf:check-export export-
name="oval:gov.nist.usgcb.windowsseven:var:22" value-
id="xccdf_gov.nist_value_password_minimum_length_var"/>
    <xccdf:check-content-ref href="USGCB-Windows-7-oval.xml"
name="oval:gov.nist.usgcb.windowsseven:def:7"/>
  </xccdf:check>
</xccdf:Rule>
```

6.7 When creating an external variable, carefully consider the possible values.

Rationale: This makes the content more readily reusable.

Applicability: This applies whenever you are creating an external variable that has several possible values, particularly if the content will be used by other parties.

Implementation: Consider the full set of possible values when creating an external variable. An example is establishing an external variable to hold a minimum password length value. Perhaps your organization has three password policies: 8, 12, and 16 character minimums. You could set the *<oval:possible_value>* element to hold 8, 12, and 16, but this precludes the use of any other policy value. So if your policy changes to a 10 character minimum, the OVAL would need to be rewritten. It might be more appropriate to use *<oval:possible_restriction>* to set a range of values and perform input validation instead of discretely defining each possible value using *<oval:possible_value>*.

If you have a variable that has an enumerated set of values, these can be specified using the *<oval:possible_value>* element as well.

Impact/Consequence: This allows a single check to be used with multiple input variables, so it reduces the number of checks that need to be written. This makes content maintenance and troubleshooting easier, and it reduces the likelihood of errors entering the content by eliminating the writing of unnecessary checks.

Example: The first example shows the use of the *<oval:possible_restriction>* element for a range of values, and the second example shows the use of the *<oval:possible_value>* element for enumerated values.

```
<external_variable comment="Required Password Length" datatype="int"
id="oval:namespace_here:var:1" version="1">
  <possible_restriction hint="Min/Max password length">
    <restriction operation="greater than or equal">0</restriction>
    <restriction operation="less than or equal">14</restriction>
  </possible_restriction>
</external_variable>
```

```
<external_variable comment="Audited events" datatype="string"
id="oval:namespace_here:var:2" version="1">
  <possible_value hint="Audit no events">AUDIT_NONE</possible_value>
  <possible_value hint="Audit success
events">AUDIT_SUCCESS</possible_value>
  <possible_value hint="Audit failure
events">AUDIT_FAILURE</possible_value>
  <possible_value hint="Audit success and failure
events">AUDIT_SUCCESS_FAILURE</possible_value>
</external_variable>
```

6.8 Reuse check content where possible.

Rationale: Reusing check content where possible reduces the likelihood of errors (typos, etc.) and makes content maintenance and troubleshooting easier.

Applicability: This applies whenever you have an opportunity to use a single object, variable, or other entity instead of duplicating the same information within multiple objects, multiple variables, etc.

Implementation: Create a single object, variable, etc. instead of duplicate objects, variables, etc. An example is having a set of checks that all look for files within the system32 directory. There should be a single object and a single variable that point to system32, and they should be reused for all the checks in the set. For example, oval:org.mitre.oval:var:200 is the ID of the system32 variable in the OVAL repository [8], and it is reused by hundreds of objects.

Impact/Consequence: This allows a single object, variable, etc. to be used with many checks, so it reduces the number of objects, variables, etc. that need to be created. This makes content maintenance and troubleshooting easier, and it reduces the likelihood of errors entering the content by eliminating the writing of unnecessary objects, variables, etc. However, be cautioned that future changes to check content should not alter the intended logic of the content, otherwise

others that use the check content may start receiving unexpected results (FALSE instead of TRUE, for example).

Example: The examples below show two *<oval:file_object>* definitions that reference the same variable in the OVAL repository, with id oval:org.mitre.oval:var:200.

```
<file_object xmlns="http://oval.mitre.org/XMLSchema/oval-definitions-
5#windows" id="oval:gov.nist.usgcb.windowsseven:obj:20003"
version="2">
  <path var_check="all" var_ref="oval:org.mitre.oval:var:200"/>
  <filename>telnet.exe</filename>
</file_object>

<file_object xmlns="http://oval.mitre.org/XMLSchema/oval-definitions-
5#windows" id="oval:gov.nist.usgcb.windowsseven:obj:20005"
version="2">
  <path var_check="all" var_ref="oval:org.mitre.oval:var:200"/>
  <filename>tftp.exe</filename>
</file_object>
```

6.9 Indicate revisions of definitions, tests, objects, states, and variables.

Rationale: Updating the version every time you revise an OVAL definition, test, object, state, or variable makes it clear that any two instances of an entity with the same version number are the same, and that any two instances of an entity with different version numbers are different.

Background: Section 4.3.3 of the OVAL Language Specification [4] defines the properties of an OVAL Definition, and they include a mandatory *<oval:version>* property that holds the version of the OVAL Definition as an unsigned integer. Although the *<oval:version>* property is mandatory, the OVAL specification and the SCAP specification do not place any requirements on the value of this property. The same is true for the *<oval:version>* properties of an OVAL Test (Section 4.3.12), OVAL Object (Section 4.3.16), OVAL State (Section 4.3.20), and OVAL Variable (Section 4.3.22).

Applicability: You want to modify an existing OVAL definition, test, object, state, or variable.

Implementation: Update the value for the *<oval:version>* property every time you are creating a new revision of an OVAL Definition, Test, Object, State, or Variable, even if you consider your changes to be minor. Ideally the values used for the *<oval:version>* property should have a sequence, such as iterative numbers (1, 2, 3, 10), so that their order can be readily determined. Tools, scripts, and other mechanisms for generating and modifying content should handle this versioning on behalf of the user.

Impact/Consequence: Clearly distinguishing each revision of an OVAL Definition, Test, Object, State, or Variable allows users to immediately tell that a new revision has been released. Users can also readily compare revision numbers to each other to determine which iteration should be used. Without clearly marking each revision, users might inadvertently fail to update

to a newer revision, or they might inadvertently confuse one revision with another. This could cause the users to get inaccurate or inconsistent results compared to other users.

Example: Below are three examples of OVAL elements with *<oval:version>* values.

```
<definition class="compliance"
id="oval:gov.nist.usgcb.windowsseven:def:1" version="2">

<registry_test xmlns="http://oval.mitre.org/XMLSchema/oval-
definitions-5#windows" check="at least one"
check_existence="at_least_one_exists" comment="Windows 7 is installed"
id="oval:org.mitre.oval:tst:10792" version="4">

<sid_object xmlns="http://oval.mitre.org/XMLSchema/oval-definitions-
5#windows" id="oval:gov.nist.usgcb.windowsseven:obj:3" version="2">
```

6.10 Have a single CCE or CVE per definition when applicable.

Rationale: Having a single identifier per definition, instead of multiple identifiers per definition, can produce more granular results.

Background: From Section 3.3 of NIST SP 800-126 Revision 2 [1]: "If an OVAL compliance class definition maps to one or more CCE identifiers, the definition SHOULD include *<oval-def:reference>* elements that reference those identifiers…" and "If an OVAL vulnerability class definition maps to one or more CVE identifiers, the definition SHOULD include *<oval-def:reference>* elements that reference those identifiers…."

Applicability: This applies to writing OVAL compliance definitions that map to CCE identifiers and OVAL vulnerability definitions that map to CVE identifiers.

Implementation: OVAL compliance and vulnerability definitions should be written granularly, so that each one applies to the fewest CCE or CVE identifiers possible, respectively. There are some cases where a single definition will map to multiple identifiers, such as multiple software flaw vulnerabilities in a single software component. However, in most cases a compliance or vulnerability definition can be written so that only a single identifier corresponds to it.

Impact/Consequence: Having more granular definitions produces more granular results. If many identifiers map to a definition, then testing for that definition simply indicates a collective result and does not indicate which identifier or identifiers are relevant for the host. This could significantly slow and complicate the process of remediating compliance issues and vulnerabilities on hosts.

Example:

```
<definition class="compliance"
id="oval:gov.nist.usgcb.windowsseven:def:1" version="2">
  <metadata>
    <title>Account Lockout Duration</title>
    <affected family="windows">
      <platform>Microsoft Windows 7</platform>
    </affected>
  <reference ref_id="CCE-9308-8" source="http://cce.mitre.org"/>
    <description>Account Lockout Duration</description>
  </metadata>
...
</definition>
```

6.11 Be careful when extending extended definitions.

Rationale: Extending an extended definition can become unnecessarily complicated, especially when there are three or more layers of extension.

Applicability: This best practice should be considered whenever a content author is contemplating extending an extended definition.

Implementation: There is nothing wrong with extending definitions, but there are concerns about extending a definition that extends a definition, and especially having even more layers of extension for definitions. This can make it very difficult to follow the flow of the XML and determine what is actually being done. Another concern is that a loop of extensions could occur (circular logic).

Impact/Consequence: Avoiding extending an extended definition, particularly with three or more layers of extension, can make content much clearer for authors, maintainers, and troubleshooters, reducing the burden on them.

6.12 Explicitly declare the *<oval:registry_state>* element's *<oval:type>* element.

Rationale: This helps ensure that registry values are handled correctly by explicitly defining their type.

Background: The *<oval:type>* element is an optional property of the *<oval:registry_state>* element. The *OVAL Language Windows Component Specification* document [5] defines it as "the type associated with the value of a hive or registry key."

Reference: For more information on the *<oval:registry_state>* element and its *<oval:type>* element, see Section 2.17 of the *OVAL Language Windows Component Specification: Version 5.10.1 Revision 1* [5].

Applicability: This is applicable whenever an *<oval:registry_state>* element is used.

Implementation: The *<oval:type>* element should be included whenever the *<oval:registry_state>* element is used to ensure that the corresponding registry values are interpreted correctly. An example is receiving the value 1: is this meant as the string "1" (reg_sz), the binary value 1 (reg_binary), or the 32-bit value 1 (reg_dword)?

Impact/Consequence: If the *<oval:type>* element is not specified, then the content author may make erroneous assumptions about the nature of the value associated with the hive or registry key. This could lead to incorrect or inconsistent results.

Example: The example below shows the use of the *<oval:type>* element within the *<oval:registry_state>* element.

```
<registry_state xmlns="http://oval.mitre.org/XMLSchema/oval-
definitions-5#windows" id="oval:gov.nist.usgcb.windowsseven:ste:2"
version="2">
  <type>reg_dword</type>
  <value datatype="int" operation="greater than or equal"
var_ref="oval:gov.nist.usgcb.windowsseven:var:2"/>
</registry_state>
```

6.13 Avoid the use of deprecated tests.

Rationale: If a test has been replaced with another test, the new test should be used in place of the deprecated test because of its superior characteristics and its continued support by the specification and tools.

Applicability: This applies whenever writing or modifying content that is based on a deprecated test.

Implementation: Instead of using a deprecated test, use the new test or tests that have replaced it. For example, it is common for a single test to be split into multiple tests to provide greater result granularity. In that case, it would be appropriate to use one or more of the new tests instead of the deprecated test.

Impact/Consequence: The assumption in the creation of a new test is that it is superior to the test or tests that it deprecates. It may offer better performance, more accurate or granular results, etc. So failing to switch to a new test may unnecessarily cause a variety of problems. Another possible consequence is that newer SCAP-validated products may not be capable of processing deprecated tests.

6.14 Ensure that the schema location and version number agree.

Rationale: Unpredictable results will occur if the schema location and version number do not agree.

Applicability: This applies whenever OVAL is being used.

Implementation: Ensure that the value assigned to the *<oval:generator>* element's *@schema_version* attribute is in agreement with the *<xsi:schemaLocation>* value. For example, don't point to the location of the OVAL 5.3 schema if you are setting the *@schema_version* attribute to 5.10.

Impact/Consequence: If the two values are not synchronized, unpredictable outcomes may occur when running the content, including tool crashes and inconsistent or inaccurate results.

Example: The examples below show the declaration of the *<xsi:schemaLocation>* element and the *<oval:schema_version>* element.

```
<oval_definitions
  ...
xsi:schemaLocation="http://oval.mitre.org/XMLSchema/oval-common-5
http://oval.mitre.org/language/version5.10/ovaldefinition/complete/oval-common-schema.xsd    http://oval.mitre.org/XMLSchema/oval-definitions-5
http://oval.mitre.org/language/version5.10/ovaldefinition/complete/oval-definitions-schema.xsd    http://oval.mitre.org/XMLSchema/oval-definitions-5#windows
http://oval.mitre.org/language/version5.10/ovaldefinition/complete/windows-definitions-schema.xsd    http://oval.mitre.org/XMLSchema/oval-definitions-5#independent
http://oval.mitre.org/language/version5.10/ovaldefinition/complete/independent-definitions-schema.xsd">

<generator>
  <oval:product_name>National Institute of Standards and Technology</oval:product_name>
  <oval:schema_version>5.10</oval:schema_version>
  <oval:timestamp>2014-02-24T10:00:00.000-04:00</oval:timestamp>
</generator>
```

7 XCCDF Style Best Practices

This section discusses style best practices specific to XCCDF.

7.1 Use a tailoring document when deriving your own XCCDF content from someone else's benchmark.

Rationale: A tailoring document allows you to customize a benchmark without directly altering the benchmark document itself.

Background: As stated in Section 6.1 of NISTIR 7275 Revision 4 [2], "A tailoring document holds exactly one `<xccdf:Tailoring>` element, which contains `<xccdf:Profile>` elements to modify the behavior of an `<xccdf:Benchmark>`." This is also referred to as the use of *external profiles*, because the profiles applied to the benchmark are external to the benchmark document.

Reference: See Section 6.7 of NISTIR 7275 Revision 4 for a more detailed explanation of tailoring documents, as well as the actual `<xccdf:Tailoring>` element specification.

Dependencies: This best practice is dependent on the best practices in Section 6.6 (Use external variables so a single check can be used for multiple input variables.) and Section 6.7 (When creating an external variable, carefully consider the possible values.)

Applicability: You want to derive your own content from an existing benchmark, such as customizing a benchmark to take into account your organization's individual needs and requirements.

Implementation: There are two options if you want to derive your own content from someone else's benchmark: directly edit the benchmark, or use a tailoring document to customize the benchmark without editing it directly. This best practice is recommending the second option over the first. You would create a tailoring document, with one or more profiles that each define a set of customizations for a single benchmark.

Impact/Consequence: As stated in Section 6.7.1 of NISTIR 7275 Revision 4, "There are several reasons why this [using a tailoring document] might be desirable:

- The benchmark might not be controlled by the organization wishing to add the profile to it.
- The benchmark might have digital signatures that would be corrupted by benchmark modification.
- The benchmark might undergo revision by its author, so modifications by a different party would represent a development fork that complicates maintenance.
- It enables the capturing of manual tailoring actions in a well-defined format...."

In summary, using a tailoring document eliminates the need to directly edit the source material. If you had the ability to directly edit the benchmark and you did so, the problems described

above would be applicable. It would be necessary to duplicate work, such as transferring customizations from one version of a benchmark to another as the benchmark is revised over time. This is error prone and time consuming. By using a tailoring document, the customizations are recorded in an efficient and consistent manner, making their transfer from one benchmark version to another trivial.

Example:

```
<Tailoring id="xccdf_gov.nist_tailoring_sample" …>
  <version time="2015-03-10T12:34:56">1</version>
  <Profile id="xccdf_gov.nist_profile_1">
    <title>Sample profile</title>
    <set-value
idref="xccdf_gov.nist_value_password_minimum_length_var" >8</set-
value>
  </Profile>
</Tailoring>
```

7.2 Indicate revisions of a single benchmark or tailoring document.

Rationale: Updating the version every time you revise an XCCDF benchmark or tailoring document makes it clear that any two instances of the document with the same version number and the same ID are the same document, and that any two instances of the document with different version numbers and the same ID are different versions of the same document.

Background: The *<xccdf:version>* element is mandatory for a benchmark document and a tailoring document. The SCAP and XCCDF specifications do not explicitly define a format for the *<xccdf:version>* element values, other than stating that the version is to be a string. See the Reference below for the benchmark recommendations.

Reference: NIST SP 800-126 Revision 2, Section 3.2.2, Item 1a: "Multiple revisions of a single benchmark SHOULD have the same *@id* attribute value and different *<xccdf:version>* element values, so that someone who reviews the revisions can readily identify them as multiple versions of a single benchmark." Item 1b: "Multiple revisions of a single benchmark SHOULD have *<xccdf:version>* element values that indicate the revision sequence, so that the history of changes from the original benchmark can be determined."

Applicability: You want to modify an existing XCCDF benchmark or tailoring document.

Implementation: Update the value for *<xccdf:version>* every time you are creating a new revision of the benchmark or tailoring document, even if you consider your changes to be minor. Ideally the values used for *<xccdf:version>* should have a sequence, such as iterative numbers (0.1, 0.2, 0.3, 1.0), so that their order can be readily determined.

Impact/Consequence: Clearly distinguishing each revision of a benchmark or tailoring document allows users of that document to immediately tell that a new revision has been released. Users can also readily compare revision numbers to each other to determine which

iteration of a document should be used. Without clearly marking each revision, users might inadvertently fail to update to a newer revision of the benchmark or tailoring document, or they might inadvertently confuse one revision with another. This could cause the users to get inaccurate or inconsistent results compared to other users.

Example: The example below shows a declaration of the *<xccdf:version>* element.

```
<xccdf:version time="2012-02-24T10:00:00"
update="http://usgcb.nist.gov">v1.2.3.1</xccdf:version>
```

7.3 Indicate revisions of <xccdf:Profile>, <xccdf:Group>, <xccdf:Rule>, and <xccdf:Value> elements.

Rationale: Updating the version every time you revise an *<xccdf:Profile>*, *<xccdf:Group>*, *<xccdf:Rule>*, or *<xccdf:Value>* elements makes it clear that any two instances of the element with the same version number and the same ID are the same element, and that any two instances of the element with different version numbers and the same ID are different versions of the same element.

Background: The *<xccdf:Profile>*, *<xccdf:Group>*, *<xccdf:Rule>*, and *<xccdf:Value>* elements all have an optional *<xccdf:version>* element intended to be used to provide a version number for the element.

Applicability: You want to modify an existing *<xccdf:Profile>*, *<xccdf:Group>*, *<xccdf:Rule>*, or *<xccdf:Value>* element.

Implementation: Update the value for *<xccdf:version>* every time you are creating a new revision of the *<xccdf:Profile>*, *<xccdf:Group>*, *<xccdf:Rule>*, or *<xccdf:Value>* element, even if you consider your changes to be minor. Ideally the values used for *<xccdf:version>* should have a sequence, such as iterative numbers (0.1, 0.2, 0.3, 1.0), so that their order can be readily determined.

Impact/Consequence: Clearly distinguishing each revision of an *<xccdf:Profile>*, *<xccdf:Group>*, *<xccdf:Rule>*, or *<xccdf:Value>* element allows users of that element to immediately tell that a new revision has been released. Users can also readily compare revision numbers to each other to determine which iteration of an element should be used. Without clearly marking each revision, users might inadvertently fail to update to a newer revision of the element, or they might inadvertently confuse one revision with another. This could cause the users to get inaccurate or inconsistent results compared to other users.

Example:

```
<xccdf:Profile id="xccdf_gov.nist_profile_1">
  <xccdf:version time="2012-02-24T10:00:00"
update="http://usgcb.nist.gov">v1.2.3.1</xccdf:version>
...
</xccdf:Profile>
```

7.4 When referencing OVAL from XCCDF, match datatypes.

Rationale: Conflicts between OVAL and XCCDF datatypes can cause unpredictable results.

Background: Table 16 in NIST SP 800-126 Revision 2, Section 3.2.5 matches OVAL variable data types to XCCDF data types (for example, OVAL int matches XCCDF number). The same section also states: "The type and value binding of the specified `<xccdf:Value>` is constrained to match that lexical representation of the indicated OVAL Variable data type. Table 16 summarizes the constraints regarding data type usage." However, there is nothing in the NIST SP that makes compliance with this matching mandatory, or even recommended.

Applicability: This is applicable whenever an OVAL variable and an XCCDF variable are in an operation together, including assignment (e.g., assigning the value of the OVAL variable to the XCCDF variable).

Implementation: OVAL and XCCDF variables in an operation together should be of compatible types. Table 16 in Section 3.2.5 of NIST SP 800-126 Revision 2 contains the definitive listing of OVAL and XCCDF variable data type mappings, which are summarized here for convenience:

- OVAL int, XCCDF number
- OVAL float, XCCDF number
- OVAL boolean, XCCDF boolean
- All other OVAL variable data types, XCCDF string

Impact/Consequence: This ensures that data being passed between OVAL and XCCDF is being used in the expected way (a number as a number, a string as a string, etc.) Failure to ensure that datatypes match can cause data passed between OVAL and XCCDF to be misused, such as attempting to misinterpret a number as a string, or a string as a number. This can cause unpredictable results.

7.5 Have a single CCE or CVE per rule when applicable.

Rationale: Having a single identifier per rule, instead of multiple identifiers per rule, can produce more granular results.

Background: From Section 3.2.4.1 of NIST SP 800-126 Revision 2 [1]: "Each `<xccdf:Rule>` element SHALL include an `<xccdf:ident>` element containing a CVE, CCE, or CPE identifier reference if an appropriate identifier exists." Note that the `<xccdf:ident>` element may be used more than one time for a single `<xccdf:Rule>` element.

Dependencies: This best practice is dependent on the best practice in Section 6.10 (Have a single CCE or CVE per definition when applicable.)

Applicability: This applies to writing `<xccdf:Rule>` elements that reference a CCE or CVE identifier.

Implementation: *<xccdf:Rule>* elements should be written granularly, so that each one applies to the fewest CCE or CVE identifiers possible. Generally this is driven by the number of identifiers used by the definition being referenced. There are some cases where a single rule will map to multiple identifiers, such as pointing to an OVAL vulnerability definition for multiple software flaw vulnerabilities in a single software component.

Impact/Consequence: Having more granular rules produces more granular results. If many identifiers map to a rule, then testing for that rule simply indicates a collective result and does not indicate which identifier or identifiers are relevant for the host. This could significantly slow and complicate the process of remediating compliance issues and vulnerabilities on hosts.

Example:

```
<xccdf:Rule id="xccdf_gov.nist_rule_account_lockout_duration"
selected="false" weight="10.0">
...
  <xccdf:ident system="http://cce.mitre.org">CCE-9308-8</xccdf:ident>
...
</xccdf:Rule>
```

7.6 If a patch checklist is required, use separate checklists for patches and configuration settings.

Rationale: Patches change at a greater rate than configuration settings, so patch content should not be integrated into configuration setting content because of their differing maintenance cycles.

Applicability: This is applicable whenever a patch checklist is required and there are also security configuration settings to be included in the checklist. This is not applicable when a patches up-to-date rule is being used, only when a full-fledged patch checklist is required.

Implementation: Create two checklists, one for the patch material and one for the configuration settings.

Impact/Consequence: If patch and configuration setting content is merged into a single checklist, then that checklist will have to be updated more frequently to keep the patch information current. This will cause new revisions of the entire checklist to be released, putting an unnecessary burden on checklist users who would have to compare the old and new checklists to determine that only the patch content has been changed. By separating the two types of content into separate checklists, users can retrieve updated copies of the patch checklist as needed without worrying about changes to the configuration checklist, which would be released separately on a less frequent schedule.

8 SCAP Data Stream Style Best Practices

This section discusses style best practices specific to SCAP data streams.

8.1 Avoid using data stream identifiers to convey other information to automated parsers.

Rationale: A data stream identifier is intended to be an identifier only and not to convey other information, such as packaging format information, so automated parsers will not know how to extract these meanings from the identifier.

Applicability: This applies whenever creating or modifying an SCAP data stream.

Implementation: Avoid including extraneous information when defining the `@id` attribute for a `<ds:data-stream>` element. An example is specifying ".zip" within the `@id` attribute value in order to indicate that the data stream has been zipped.

Impact/Consequence: If parsing a data stream is dependent on automatically extracting additional values from within the `@id` attribute, this is likely to fail for many parsers, preventing the reading and processing of the data stream. Relying on this method even with parsers that support it may produce unpredictable results because of the nature of data streams. For example, suppose that the zipped nature of a data stream is indicated by including ".zip" in the `@id` attribute. If that data stream is unzipped, there is no mechanism for updating that `@id` attribute's value to indicate that the data stream is no longer zipped.

9 Best Practice Topics for Further Discussion

This section details potential best practice topics where the authors feel that community feedback is needed before further developing the best practice. This section will only be included in the public comment draft, not the final version of the publication.

9.1 Is it preferable to use plaintext or XHTML?

Rationale: Plaintext supports greater interoperability but Extensible Hypertext Markup Language (XHTML) gives content authors the ability to specify style for human readability.

Applicability: This applies to all SCAP elements that support XHTML.

Implementation: Plaintext supports interoperability because some tools are not presenting XHTML, which is causing XHTML content to be stripped out. If structural markup is used in XHTML, its textual elements can easily be transformed to other formats, negating the need to display XHTML. XHTML gives content authors much greater control over how their content is visually presented to users, unlike plaintext, which provides no control.

Impact/Consequence: Requiring the use of plaintext over XHTML would take away style control from content authors while improving interoperability. Requiring the use of structural markup whenever using XHTML would remedy the problem somewhat, but not completely because of lack of tool support. Requiring the use of XHTML would make the creation of simple content overly complicated.

Appendix A—Acronyms and Abbreviations

Selected acronyms and abbreviations used in this paper are defined below.

CCE	Common Configuration Enumeration
CPE	Common Platform Enumeration
CVE	Common Vulnerabilities and Exposures
FISMA	Federal Information Security Management Act
IR	Internal Report
ITL	Information Technology Laboratory
NIST	National Institute of Standards and Technology
NISTIR	National Institute of Standards and Technology Internal Report
OCIL	Open Checklist Interactive Language
OMB	Office of Management and Budget
OVAL	Open Vulnerability and Assessment Language
RFC	Request for Comments
SCAP	Security Content Automation Protocol
SP	Special Publication
TMSAD	Trust Model for Security Automation Data
USGCB	United States Government Configuration Baseline
XCCDF	Extensible Configuration Checklist Description Format
XHTML	Extensible Hypertext Markup Language

Appendix B—References

[1] NIST, NIST SP 800-126 Revision 2, *The Technical Specification for the Security Content Automation Protocol (SCAP): SCAP Version 1.2*, September 2011. http://csrc.nist.gov/publications/nistpubs/800-126-rev2/SP800-126r2.pdf and errata (http://csrc.nist.gov/publications/nistpubs/800-126-rev2/sp800-126r2-errata-20120409.pdf)

[2] NIST, NISTIR 7275 Revision 4, *Specification for the Extensible Configuration Checklist Description Format (XCCDF) Version 1.2*, September 2011. http://csrc.nist.gov/publications/nistir/ir7275-rev4/nistir-7275r4_updated-march-2012_clean.pdf

[3] NIST, NISTIR 7692, *Specification for the Open Checklist Interface Language (OCIL) Version 2.0*, April 2011. http://csrc.nist.gov/publications/nistir/ir7692/nistir-7692.pdf

[4] The MITRE Corporation, *The OVAL Language Specification, Version 5.10.1*, January 2012. https://oval.mitre.org/language/version5.10.1/OVAL_Language_Specification_01-20-2012.pdf

[5] The MITRE Corporation, *The OVAL Language Windows Component Specification: Version 5.10.1 Revision 1*, January 2012. https://oval.mitre.org/language/version5.10.1/OVAL_Windows_Component_Specification_01-19-2012.pdf

[6] W3C, *XML Schema*. http://www.w3.org/XML/Schema.html

[7] NIST, NISTIR 7802, *Trust Model for Security Automation Data 1.0 (TMSAD)*, September 2011. http://csrc.nist.gov/publications/nistir/ir7802/NISTIR-7802.pdf

[8] The MITRE Corporation, OVAL Repository. https://oval.mitre.org/repository/

[9] Kent Landfield, McAfee, "Content Development Best Practices" presentation. http://scap.nist.gov/events/2011/saddsp/presentations/Kent_Landfield-Content_Best_Practices.pdf

[10] Shane Shaffer, G2, Inc., "Content Development Best Practices" presentation, http://makingsecuritymeasurable.mitre.org/participation/devdays.html#2011

[11] USGCB Checklist for Windows 7, https://web.nvd.nist.gov/view/ncp/repository/checklistDetail?id=295