National Institute of Standards and Technology

U.S. Department of Commerce

NIST Interagency Report 7275
Revision 4 (DRAFT)

Specification for the Extensible Configuration Checklist Description Format (XCCDF) Version 1.2 (DRAFT)

Neal Ziring
David Waltermire

NIST Interagency Report 7275
Revision 4 (DRAFT)

Specification for the Extensible Configuration Checklist Description Format (XCCDF) Version 1.2 (DRAFT)

Neal Ziring
David Waltermire

C O M P U T E R S E C U R I T Y

Computer Security Division
Information Technology Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899-8930

July 2010

U.S. Department of Commerce

Gary Locke, Secretary

National Institute of Standards and Technology

Dr. Patrick D. Gallagher, Director

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 analysis to advance the development and productive use of information technology. ITL's responsibilities include the development of technical, physical, administrative, and management standards and guidelines for the cost-effective security and privacy of sensitive unclassified information in Federal computer systems. This Interagency Report discusses ITL's research, guidance, and outreach efforts in computer security and its collaborative activities with industry, government, and academic organizations.

National Institute of Standards and Technology Interagency Report 7275 Revision 4
142 pages (July 2010)

Abstract

This report specifies the data model and Extensible Markup Language (XML) representation for the Extensible Configuration Checklist Description Format (XCCDF) Version 1.2. An XCCDF document is a structured collection of security configuration rules for some set of target systems. The XCCDF specification is designed to support information interchange, document generation, organizational and situational tailoring, automated compliance testing, and compliance scoring. The specification also defines a data model and format for storing results of security guidance or checklist compliance testing. The intent of XCCDF is to provide a uniform foundation for expression of security checklists and other configuration guidance, and thereby foster more widespread application of good security practices.

Purpose and Scope

The XCCDF standardized XML format enables an automated provisioning of recommendations for minimum security controls for information systems categorized in accordance with NIST Special Publication (SP) 800-53, *Recommended Security Controls for Federal Information Systems,* and Federal Information Processing Standards (FIPS) 199, *Standards for Security Categorization of Federal Information and Information Systems*, to support Federal Information Security Management Act (FISMA) compliance efforts.

To promote the use, standardization, and sharing of effective security checklists, NIST and the National Security Agency (NSA) have collaborated with representatives of private industry to develop the XCCDF specification. The specification is vendor-neutral, flexible, and suited for a wide variety of checklist applications.

Audience

The primary audience of the XCCDF specification is government and industry security analysts, and industry security management product developers. NIST and NSA welcome feedback from these groups on improving the XCCDF specification.

Table of Contents

Acknowledgements

The authors of this report, Neal Ziring of the National Security Agency (NSA) and David Waltermire of the National Institute of Standards and Technology (NIST), would like to acknowledge the following individuals who contributed to the initial definition and development of the Extensible Configuration Checklist Description Format (XCCDF): David Proulx, Mike Michnikov, Andrew Buttner, Todd Wittbold, Adam Compton, George Jones, Chris Calabrese, John Banghart, Murugiah Souppaya, John Wack, Trent Pitsenbarger, and Robert Stafford. Stephen D. Quinn, Peter Mell, Matthew Wojcik, and Karen Scarfone contributed to Revisions 1, 2, and 3 of this report. Charles Schmidt was instrumental in supporting the development of XCCDF through his moderation of XCCDF working sessions and the email list; he contributed many important concepts and constructs to revision 4 of this report. Ryan Wilson of Georgia Institute of Technology also made substantial contributions. Thanks also go to the Defense Information Systems Agency (DISA) Field Security Office (FSO) Vulnerability Management System (VMS)/Gold Disk team for extensive review and many suggestions.

Trademark Information

Warnings

1. Introduction

The Extensible Configuration Checklist Description Format (XCCDF) was originally intended to be used for technical security checklists. Although this is still the primary use of XCCDF, XCCDF also has extensions into non-technical applications (e.g., owner's manuals, user guides, non-technical Federal Information Security Management Act [FISMA] controls, and items considered "manual procedures").

The security of an information technology (IT) system typically can be improved if the identified software flaws and configuration settings that affect security are properly addressed. The security of an IT system may be measured in a variety of ways; one operationally relevant method is determining conformance of the system configuration to a specified security baseline, guidance document, or checklist. These typically include criteria and rules for hardening a system against the most common forms of compromise and exploitation, and for reducing the exposed "attack surface" of a system. Many companies, government agencies, community groups, and product vendors generate and disseminate security guidance documents. While definition of the conditions under which a security setting should be applied can differ among the guidance documents, the underlying specification, test, and report formats used to identify and remediate said settings tend to be specialized and unique.

Configuring a system to conform to specified security guidance (e.g., NIST Special Publication [SP] 800-68, *Guidance for Securing Microsoft Windows XP Systems for IT Professionals: A NIST Security Configuration Checklist*, any of the Defense Information Systems Agency [DISA] Secure Technology Implementation Guides [STIG] and subsequent checklists) or other security specification is a highly technical task. To aid system administrators, commercial and community developers have created automated tools that can both determine a system's conformance to a specified guideline and provide or implement remediation measures. Many of these tools are data-driven in that they accept a security guidance specification in some program-readable form (e.g., XML, .inf, csv), and use it to perform the checks and tests necessary to measure conformance, generate reports, and perhaps remediate as directed. However, with rare exceptions, none of these tools (commercial or government developed) employ the same data formats. This unfortunate situation perpetuates a massive duplication of effort among security guidance providers and provides a barrier for content and report interoperability.

This report describes a standard data model and processing discipline for supporting secure configuration and assessment. The requirements and goals are explained in the main content; however, in summary, this report addresses:

- Document generation
- Expression of policy-aware configuration rules
- Support for conditionally applicable, complex, and compound rules
- Support for compliance report generation and scoring
- Support for customization and tailoring.

The model and its XML representation are intended to be platform-independent and portable, to foster broad adoption and sharing of rules. The processing discipline of the format requires, for some uses, a service layer that can collect and store system information and perform simple

policy-neutral tests against the system information; this is true for technical and non-technical applications of XCCDF. These conditions are described in detail below. The XML representation is expressed as an XML Schema in Appendix A.

1.1. Background

Today, groups promoting good security practices and system owners wishing to adopt them face an increasingly large and complex problem. As the number of IT systems increases, automated tools are necessary for uniform application of security rules and visibility into system status. These conditions have created a need for mechanisms that:

- Ensure compliance to multiple policies (e.g., IT Systems subject to FISMA, STIG, and/or Health Insurance Portability and Accountability Act [HIPAA] compliance)
- Permit faster, more cooperative, and more automated definition of security rules, procedures, guidance documents, alerts, advisories, and remediation measures
- Permit fast, uniform, manageable administration of security checks and audits
- Permit composition of security rules and tests from different community groups and vendors
- Permit scoring, reporting, and tracking of security status and checklist conformance, both over distributed systems and over the same systems across their operational lifetimes
- Foster development of interoperable community and commercial tools for creating and employing security guidance and checklist data.

Today, such mechanisms exist only in some isolated niche areas (e.g., Microsoft Windows patch validation) and they support only narrow slices of security guidance compliance functionality. For example, patch checking and secure configuration guidance often are not addressed at the same level of detail (or at all) in a single guidance document; however, both are required to secure a system against known attacks. This specification report proposes a data model and format specification for an extensible, interoperable checklist language that is capable of including both technical and non-technical requirements in the same XML document.

1.2. Vision for Use

XCCDF is designed to enable easier, more uniform creation of security checklists and procedural documents, and allow them to be used with a variety of commercial, Government off-the-shelf (GOTS), and open source tools. The motivation for this is improvement of security for IT systems, including the Internet, by better application of known security practices and configuration settings.

One potential use for XCCDF is streamlining compliance to FISMA and Department of Defense (DOD) STIGs. Federal agencies, state and local governments, and the private sector have difficulty measuring the security of their IT systems. They also struggle to both implement technical policy (e.g., DISA STIGs, NIST SPs) and then to demonstrate unambiguously to various audiences (e.g., Inspector General, auditor) that they have complied and ultimately improved the security of their systems. This difficulty arises from various causes, such as different interpretations of policy, system complexity, and human error. XCCDF proposes to automate certain technical aspects of security by converting English text contained in various

publications (e.g., configuration guides, checklists, the National Vulnerability Database [NVD]) into a machine-readable XML format such that the various audiences (e.g., scanning vendors, checklist/configuration guide, auditors) will be operating in the same semantic context. The end result will allow organizations to use commercial off-the-shelf (COTS) tools to automatically check their security and map to technical compliance requirements.

The scenarios below illustrate some uses of security checklists and tools that XCCDF fosters.

Scenario 1 –
An academic group produces a checklist for secure configuration of a particular server operating system version. A government organization issues a set of rules extending the academic checklist to meet more stringent user authorization criteria imposed by statute. A medical enterprise downloads both the academic checklist and the government extension, tailors the combination to fit their internal security policy, and applies an enterprise-wide audit using a commercial security audit tool. Reports output by the tool include remediation measures which the medical enterprise IT staff can use to bring their systems into full internal policy compliance.

Scenario 2 –
A federally-funded lab issues a security advisory about a new Internet worm. In addition to a prose description of the worm's attack vector, the advisory includes a set of short checklists in a standard format that assess vulnerability to the worm for various operating system platforms. Organizations all over the world pick up the advisory, and use installed tools that support the standard format to check their status and fix vulnerable systems.

Scenario 3 –
An industry consortium, in conjunction with a product vendor, wants to produce a security checklist for a popular commercial server. The core security settings are the same for all OS platforms on which the server runs, but a few settings are OS-specific. The consortium crafts one checklist in a standard format for the core settings, and then writes several OS-specific ones that incorporate the core settings by reference. Users download the core checklist and the OS-specific extensions that apply to their installations, and then run a checking tool to score their compliance with the checklist.

1.3. Summary of Changes since Version 1.0

XCCDF 1.0 received some review and critique after its release in January 2005. Most of the additions and changes come directly from suggestions by users and potential users. Other changes have been driven by the evolution of the NIST Security Content Automation Protocol (SCAP) initiatives. The list below describes the major changes; other differences are noted in the text.

- Persistent/standard identifiers - To foster standardization and re-use of XCCDF rules, community members suggested that Rule objects bear long-term, globally unique identifiers. Support for identifiers, along with the scheme or organization that assigns them, is now part of the Rule object.

- Versioning - To foster re-use of XCCDF rules, and to allow more precise tracking of Benchmark results over time, Benchmarks, Rules, and Profiles all support a version number. The version number also now supports a timestamp.

- Severity - Rules can now support a severity level: info, low, medium, and high. Severity levels can be adjusted via Profiles.

- Signatures – To foster sharing of XCCDF rules and groups of rules, each object that can be a standalone XCCDF document can have an XML digital signature: Benchmark, Group, Rule, Value, Profile, and TestResult. This allows any shared XCCDF object to have integrity and authenticity assurance.

- Rule result enhancements – Recording Benchmark results has been improved in version 1.1: the 'override' property was added for rule-results in a TestResult object, several new Rule result status values have been added, and better instance detail support was added to rule-results for multiply-instantiated Rules. Also, the descriptions of the different Rule result status values and their role in scores have been clarified.

- Enhancements for remediation - Several minor enhancements were made to the Rule's properties for automated and interactive remediation (the Rule object's 'fix' and 'fixtext' elements).

- Interactive Value tailoring – To foster interactive tailoring by tools that can support it, the 'interactive' property was added to Value objects. It gives a Benchmark checking tool a hint that it should solicit a new value prior to each application of the Benchmark. Also, the 'interfaceHint' property was added, to allow a Benchmark author to suggest a UI model to the tool.

- Scoring models – XCCDF 1.0 had only a single scoring model. 1.1 supports the notion of multiple scoring models, and two new models have been added to the specification. To support custom scoring models, the model and param properties have been added to the TestResult's score element.

- Re-usable plain text blocks – To foster re-use of common text with a Benchmark document, version 1.1 now supports definition of named, re-usable text blocks.

- Richer XHTML references – Formatted text within XCCDF Benchmarks can now use XHTML object and anchor tags to reference other XCCDF objects within a generated document.

- Target facts – It is important for a Benchmark test result to include extensive information from the system that was tested. To support this, the TestResult object now supports a list of target facts. Tools can use this list to store any relevant information they collect about the target platform or system.

- Complex checks – The Rule object now supports a mechanism for using Boolean operators to compose complex checks from multiple individual checks.

- Extension control – To give Benchmark authors more control over XCCDF inheritance, version 1.1 was enhanced to support the 'override' attribute on most property element children that can appear more than once in a Rule, Group, Value, or Profile.

- Value acquisition support – The new 'source' property on the Value object allows a Benchmark author to suggest one or more possible ways to obtain correct or candidate values. The suggestions must be given as URIs.

- Profile notes – To support better descriptions for Profiles, version 1.1 was extended to support a descriptive note facility for relating Rules and Profiles.

- Alternate check content – In 1.1.3, the semantics of checks permit multiple (alternative) references to checking engine content.

- Multiple alternative requirements – In 1.1.3, the requires property of Items has been extended to allow specification of several alternative Items, any one of which satisfies the requirement.

- Import from checking system – In 1.1.4, the check element has been extended to allow a benchmark author to specify values to retrieve from the checking system.

- Profile enhancements – In 1.1.4, selectors in Profiles may contain remarks. Also, the semantics of Profile operation have been clarified.

- Weight reporting – For 1.1.4, the weight attribute was added to Rule result elements, to allow the weight used for scoring to recorded as part of the test result object.

- CPE compatibility – Applicability of XCCDF Rules and other objects to specific IT platforms may be specified using Common Platform Enumeration (CPE) identifiers. XCCDF 1.1.4 mandates use of CPE version 2.0. All prior platform identifier support is deprecated. XCCDF 1.2 amended this to require CPE version 2.3.

- Benchmark styles – Two attributes were added to the Benchmark object in 1.1.4, to allow optional specification of a Benchmark style.

- TestResult enhancement – Two properties were added to the TestResult object to allow recording the responsible organization and the system identity under which the results were obtained.

- Absolute scoring model – This new model gives a score of 1 when the target passes all applicable rules, and 0 otherwise.

- Impact metric – For 1.1.4, the impact-metric element was added to associate an impact statement with a Rule. Use of this field was deprecated in 1.2.

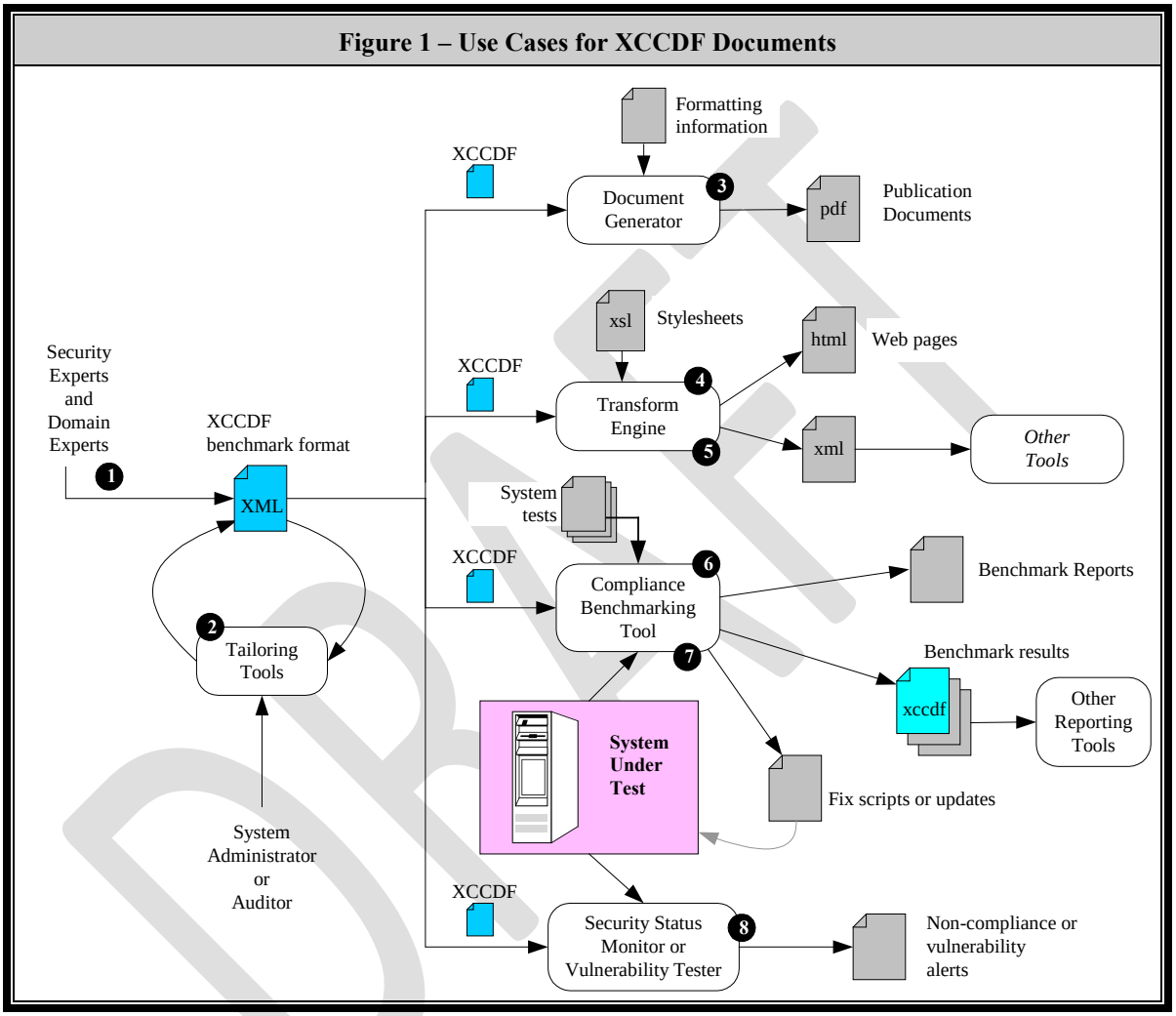
- Metadata – In release 1.2 the flexibility of the metadata field was greatly expanded and metadata fields were added to all major XCCDF structures.

- Check import – For 1.2, the use of the check-import property was clarified and the import-xpath attribute was added to further control import actions.
- Text classification – The XCCDF 1.2 specification defines some sample classes to support stylistic labeling of XHTML content.
- Complex values – XCCDF 1.2 adds the concept of a complex value capable of holding lists and/or externally defined data types.
- Group extension – Group extension is deprecated in 1.2.
- Selector processing – The processing of Profile selectors in 1.2 now explicitly permits selectors to have overlapping scopes.
- DC status – In 1.2 the dc-status field has been added to store status information in Dublin Core format.
- Role property – Use of the role property is deprecated in 1.2.
- Check negation – Results of checks can now be negated in 1.2.

2. Requirements

The general objective for XCCDF is to allow security analysts and IT experts to create effective and interoperable automated checklists, and to support the use of automated checklists with a wide variety of tools. Figure 1 shows some potential utilization scenarios.

Figure 1 – Use Cases for XCCDF Documents



The following list describes some requirements for several of the use cases depicted in Figure 1:

1. Security and domain experts create a security guidance document or checklist, which is an organized collection of rules about a particular kind of system or platform. To support this use, XCCDF must be an open, standardized format, amenable to generation by and editing with a variety of tools. It must be expressive enough to represent complex conditions and relationships about the systems to be assessed, and it must also incorporate descriptive material and remediative measures. (XCCDF Benchmarks may include specification of the hardware and/or software platforms to which they apply; however, it is recommended that programmatically ascertainable information should be relegated to the lower-level identification and remediation languages. For specifying programmatically ascertainable information in the XCCDF file, the specification should

be concrete and granular so that compliance checking tools can detect if a Rule is suited for a target platform.)

2. Auditors and system administrators may employ tailoring tools to customize a security guidance document or checklist for their local environment or policies. For example, although NIST produces the technical security guidance for Windows XP Professional in the form of Special Publication 800-68, certain Federal agencies may have trouble applying all settings without exception. For those settings which hinder functionality (perhaps with Legacy systems, or in a hybrid Windows 2000/2003 domain), the agency may wish to tailor the XML received from the NIST Web site. For this reason, an XCCDF document must include the structure and interrogative text required to direct the user through the tailoring of a Benchmark, and it must be able to hold or incorporate the user's tailoring responses. For example, a checklist user might want to set the password policy requirement to be more or less stringent than the provided security recommendations. XCCDF should be extensible to allow for the custom tailoring and inclusion of the explanatory text for deviation from recommended policy.
3. Although the goal of XCCDF is to distill English (or other language) prose checklists into machine-readable XML, XCCDF should be structured to foster the generation of readable prose documents from XCCDF-format documents.
4. The structure of an XCCDF document should support transformation into HTML, for posting the security guidance as a Web page.
5. An XCCDF document should be transformable into other XML formats, to promote portability and interoperability.
6. The primary use case for an XCCDF-formatted security guidance document is to facilitate the normalization of configuration content through automated security tools. Such tools should accept one or more XCCDF documents along with supporting system test definitions, and determine whether or not the specified rules are satisfied by a target system. The XCCDF document should support generation of a compliance report, including a weighted compliance score.
7. In addition to a report, some tools may utilize XCCDF-formatted content (and associated content from other tools) to bring a system into compliance through the remediation of identified vulnerabilities or misconfigurations. XCCDF must be able to encapsulate the remediation scripts or texts, including several alternatives.
8. XCCDF documents might also be used in vulnerability scanners, to test whether or not a target system is vulnerable to a particular kind of attack. For this purpose, the XCCDF document would play the role of a vulnerability alert, but with the ability to both describe the problem and drive the automated verification of its presence.

In addition to these use cases, an XCCDF document should be amenable to embedding inside other documents. Likewise, XCCDF's extensibility should include provisions for incorporating other data formats. And finally, XCCDF must be extensible to include new functionality, features, and data stores without hindering the functionality of existing XCCDF-capable tools.

2.1. Structure and Tailoring Requirements

The basic unit of structure for a security guidance document or checklist is a *rule*. A rule simply describes a state or condition which the target of the document should exhibit. A simple document might consist of a simple list of rules, but richer ones require additional structure.

To support customization of the standardized XML format and subsequent generation of documents by and for consumers, XCCDF must allow authors to impose organization within the document. One such basic requirement is that authors will need to put related rules into named groups.

An author must be able to designate the order in which rules or groups are processed. In the simplest case, rules and groups are processed in sequence according to their location in the XCCDF document.

To facilitate customization, the author should include descriptive and interrogative text to help a user make tailoring decisions. The following two customization options are available:

> *Selectability* – A tailoring action might select or deselect a rule or group of rules for inclusion or exclusion from the security guidance document. For example, an entire group of rules that relate to physical security might not apply if one were conducting a network scan. In this case, the group of rules delineated under the physical security group could be deselected. In the case of NIST SP 800-53, certain rules apply according to the Impact Rating of the system. For this purpose, systems that have an Impact Rating of *Low* might not have all of the same access control requirements as a system with a *High* Impact Rating, and therefore the those rules that are not applicable for the *Low* system can be deselected.
>
> *Substitution* – A tailoring action might substitute a locally-significant value for a general value in a rule. For example, at a site where all logs are sent to a designated logging host, the address of that log server might be substituted into a rule about audit configuration. Using the NIST SP 800-53 example, a system with an Impact Rating of *High* might require a 12-character password, whereas a system with an Impact Rating of *Moderate* might only require an 8-character password. Depending on the Impact Rating of the target system, the user can customize or tailor the value through substitution.

When customizing security guidance documents, the possibility arises that some rules within the same document might conflict or be mutually exclusive. To avert potential problems, the author of a security guidance document must be able to identify particular tailoring choices as *incompatible*, so that tailoring tools can take appropriate actions.

In addition to specifying rules, XCCDF supports structures that foster use and re-use of rules. To this end, XCCDF must provide a means for related rules to be grouped and for sets of rules and groups to be designated, named, and easily applied. Examples of this requirement are demonstrated by DISA's Gold and Platinum distinction with respect to STIG compliance (Gold being the less stringent of the two levels). NIST also provides distinctions according to environment and Impact Rating (High, Moderate, or Low) [13]. Likewise, the Center for Internet Security (CIS) designates multiple numeric levels for their checklists (e.g., Level 1, Level 2).

To facilitate XCCDF adoption for the aforementioned requirements, XCCDF provides two basic processing modes: rule checking and document generation. It must be possible for a security

guidance or checklist author to designate the modes (e.g., Gold, Platinum, High Impact Rating, Level 1) under which a rule should be processed.

2.2. Inheritance and Inclusion Requirements

Some use cases require XCCDF to support mechanisms for authors to extend (inherit from) existing rules, in addition to expressing rules in their entirety. For example, it must be possible for one XCCDF document to include all or part of another as demonstrated in the following scenarios:

- An organization might choose to define a foundational XCCDF document for a family of platforms (e.g., Unix-like operating systems) and then extend it for specific members of the family (e.g., Solaris) or for specific roles (e.g., mail server).
- An analyst might choose to make an extended version of an XCCDF document by adding new rules and adjusting others.
- If the sets of rules that constitute an XCCDF document come from several sources, it is useful to aggregate them using an inclusion mechanism. (Note: The XCCDF specification does not define its own mechanisms for inclusion; instead, implementations of XCCDF tools should support the XML Inclusion (XInclude) facility standardized by the World Wide Web Consortium [W3C] [11].)
- Within an XCCDF document, it is desirable to share descriptive material among several rules, and to allow a specialized rule to be created by extending a base rule.
- For updating an XCCDF document, it is convenient to incorporate changes or additions using extensions.
- To allow broader site-specific or enterprise-specific customization, a user might wish to override or amend any portion of an XCCDF rule.

2.3. Document and Report Formatting Requirements

Generating English (or other language) prose documents from the underlying XCCDF constitutes a primary use case. Authors require mechanisms for formatting text, including images, and referencing other information resources. These mechanisms must be separable from the text itself, so each can be filtered out by applications that do not support or require them. (XCCDF 1.2 currently satisfies these formatting requirements mainly by allowing inclusion of Extensible Hypertext Markup Language [XHTML] markup tags [4].)

The XCCDF language must also allow for the inclusion of content that does not contribute directly to the technical content. For example, authors tend to include ‘front matter’ such as an introduction, a rationale, warnings, and references. XCCDF allows for the inclusion of intra-document and external references and links.

2.4. Rule Checking Requirements

One of XCCDF’s main features is the organization and selection of target-applicable groups and rules for performing security and operational checks on systems. Therefore, XCCDF must have access to granular and expressive mechanisms for checking the state of a system according to the rule criteria. The model for this requirement includes the notion of collecting or acquiring the state of a target system, and then checking the state for conformance to conditions and criteria

expressed as rules. The operations used have varied with different existing applications; some rule checking systems use a database query operation model, others use a pattern-matching model, and still others load and store state in memory during execution. Rule checking mechanisms used for XCCDF must satisfy the following criteria:

- **The mechanism must be able to express both positive and negative criteria.** A positive criterion means that if certain conditions are met, then the system satisfies the check, while a negative criterion means that if the conditions are met, the system fails the check. Experience has shown that both kinds are necessary when crafting criteria for checks.
- **The mechanism must be able to express Boolean combinations of criteria.** It is often impossible to express a high-level security property as a single quantitative or qualitative statement about a system's state. Therefore, the ability to combine statements with 'and' and 'or' is critical.
- **The mechanism must be able to incorporate tailoring values set by the user.** As described above, substitution is important for XCCDF document tailoring. Any XCCDF checking mechanism must support substitution of tailored values into its criteria or statements as well as tailoring of the selected set of rules.

A single rule specification scheme (e.g., Open Vulnerability and Assessment Language [OVAL] [16]) may not satisfy all potential uses of XCCDF. To facilitate other lower-level rule checking systems, XCCDF supports referencing by including the appropriate file and check reference in the XCCDF document. It is important that the rule checking system be defined separately from XCCDF itself, so that both XCCDF and the rule checking system can evolve and be used independently. This duality implies the need for a clear interface definition between XCCDF and the rule checking system, including the specification of how information should pass from XCCDF to the checking system and vice versa.

2.5. Test Results Requirements

Another objective of XCCDF is to facilitate a standardized reporting format for automated tools. In the case of many Federal agencies, several COTS and GOTS products are used to determine the security of IT systems and their compliance to various stated polices. Unfortunately, the outputs from these tools are not standardized and therefore costly customization can be required for trending, aggregation, and reporting. Addressing this use case, XCCDF provides a means for storing the results of the rule checking subsystem.

Security tools sometimes include only the results of the test or tests in the form of a pass/fail status. Other tools provide additional information so that the user does not have to access the system to determine additional information (e.g., instead of simply indicating that more than one privileged account exists on a system, certain tools also provide the list of privileged accounts). Independent of the robust or minimal reporting of the checking subsystem, the following information is basic to all results:

- The security guidance document or checklist used, along with any adaptations via customization or tailoring applied
- Information about the target system to which the test was applied, including arbitrary identification and configuration information about the target system

- The time interval of the test, and the time instant at which each individual rule was evaluated
- One or more compliance scores
- References to lower-level details possibly stored in other output files.

2.6. Metadata and Security Requirements

As the recognized need for security increases, so does the number of recommended security configuration guidance documents. The DISA STIGs and accompanying checklist documents have been available under http://iase.disa.mil/stigs for many years. Likewise, NIST's interactions with vendors and agencies have yielded checklist content provided at http://checklists.nist.gov/[1]. NSA maintains a web site offering security guidance at http://www.nsa.gov/snac/, and CIS provides checklist content at http://cisecurity.org/. Likewise, product vendors such as Microsoft Corporation, Sun Microsystems, Apple Computer, and Hewlett-Packard (to name a few) are providing their own security guidance documents independent of traditional user guides.

As of late 2007, the majority of these checklists exist in various repositories in English prose format; however, there is a recognized need and subsequent migration effort to represent said checklists in standardized XML format. To facilitate discovery and retrieval of security guidance documents in repositories and on the open Internet, XCCDF must support inclusion of metadata about a document. Some of the metadata that must be supported include: title, name of author(s), organization providing the guidance, version number, release date, update URL, and a description. Since a number of metadata standards already exist, it is preferable that XCCDF simply incorporate one or more of them rather than defining its own metadata model.

In addition to specifying rules to which a target system should comply, an XCCDF document must support mechanisms for describing the steps to bring the target into compliance. While checking compliance to a given security baseline document is common, remediation of an IT system to the recommended security baseline document should be a carefully planned and implemented process. Security guidance users should be able to trust security guidance documents, especially if they intend to accept remediation advice from them. Therefore, XCCDF must support a mechanism whereby guidance users can validate the integrity, origin, and authenticity of guidance documents.

Digital signatures are the natural mechanism to satisfy these integrity and proof-of-origin requirements. Fortunately, mature standards for digital signatures already exist that are suitable for asserting the authorship and protecting the integrity of guidance documents. XCCDF provides a means to hold such signatures, and a uniform method for applying and validating them.

[1] The National Checklist Program website is based on the NIST SP 800-70 Revision 1; *National Checklist Program for IT Products--Guidelines for Checklist Users and Developers*.

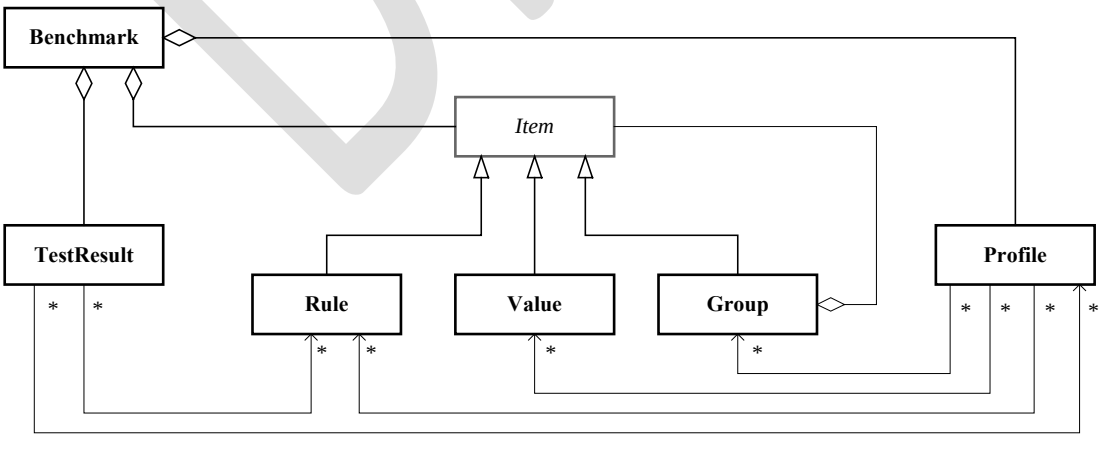
3. Data Model

The fundamental data model for XCCDF consists of four main object data types:

1. **Benchmark.** An XCCDF document holds exactly one Benchmark object. A Benchmark holds descriptive text, and acts as a container for Items and other objects.
2. **Item.** An Item is a named constituent of a Benchmark; it has properties for descriptive text, and can be referenced by an id. There are several derived classes of Items:
   - **Group.** This kind of Item can hold other Items. A Group may be selected or unselected. (If a Group is unselected, then all of the Items it contains are implicitly unselected.)
   - **Rule.** This kind of Item holds check references, a scoring weight, and may also hold remediation information. A Rule may be selected or unselected.
   - **Value.** This kind of Item is a named data value that can be substituted into other Items' properties or into checks. It can have an associated data type and metadata that express how the value should be used and how it can be tailored.
3. **Profile.** A Profile is a collection of attributed references to Rule, Group, and Value objects. It supports the requirement to allow definition of named levels or baselines in a Benchmark (see Section 2.1).
4. **TestResult.** A TestResult object holds the results of performing a compliance test against a single target device or system.

Figure 2 shows the data model relationships as a Unified Modeling Language (UML) diagram. As shown in the figure, one Benchmark can hold many Items, but each Item belongs to exactly one Benchmark. Similarly, a Group can hold many Items, but an Item may belong to only one Group. Thus, the Items in an XCCDF document form a tree, where the root node is the Benchmark, interior nodes are Groups, and the leaves are Values and Rules.

Figure 2 –XCCDF High-Level Data Model



A Profile object references Rule, Value, and Group objects. A TestResult object references Rule and Value objects and may also reference a Profile object.

The definition of a Value or Rule can extend another Value or Rule.. The extending Item inherits property values from the extended Item. This extension mechanism is separate and independent of grouping.

Group and Rule items can be marked by a Benchmark author as selected or unselected. A Group or Rule that is not selected does not undergo processing. The author may also stipulate, for a Group, Rule, or Value, whether or not the end user is permitted to tailor it.

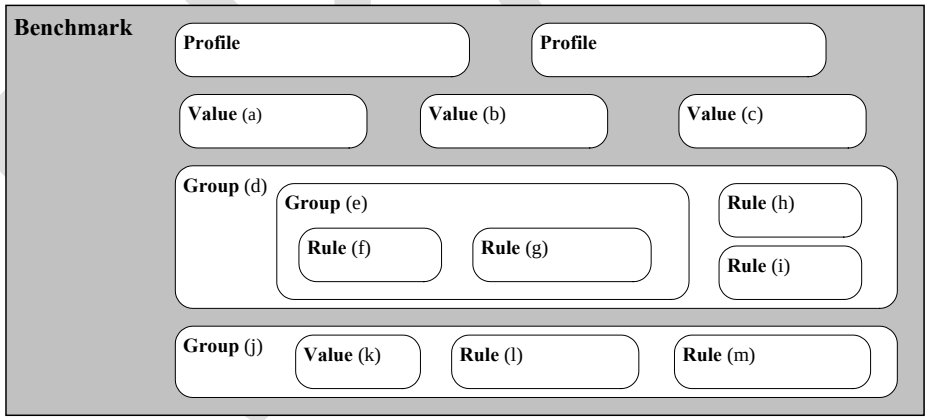
Rule and Group items may have a scoring weight associated with them, which can be used by a Benchmark checking tool to compute a target system's overall compliance score. Rule items may also hold remediation information.

Value items include information about current, default, and permissible values for the Value. Each of these properties of a Value can have an associated selector id, which is used when customizing the Value as part of a Profile. For example, a Value might be used to hold a Benchmark's lower limit for password length on some operating system. In a Profile for that operating system to be used in a closed lab, the default value might be 8, but in a Profile for that operating system to be used on the Internet, the default value might be 12.

3.1. Benchmark Structure

Typically, a Benchmark would hold one or more Groups, and each group would hold some Rules, Values, and additional child Groups. Figure 3 illustrates this relationship.

Figure 3 – Typical Structure of a Benchmark



Groups allow a Benchmark author to collect related Rules and Values into a common structure and provide descriptive text and references about them. Further, groups allow Benchmark users to select and deselect related Rules together, helping to ensure commonality among users of the same Benchmark. Lastly, groups affect Benchmark compliance scoring. As Section 3.3 explains, an XCCDF compliance score may be calculated for each group, based on the Rules and Groups in it. In such a scoring model, the overall XCCDF score for the Benchmark is computed only from the scores on the immediate Group and Rule children of the Benchmark object. In the

tiny Benchmark shown in Figure 3, the Benchmark score would be computed from the scores of Group (d) and Group (j). The score for Group (j) would be computed from Rule (l) and Rule (m).

Inheritance

The possible inheritance relations between Item object instances are constrained by the tree structure of the Benchmark, but are otherwise independent of it. In other words, all extension relationships must be resolved before the Benchmark can be used for compliance testing. An Item may only extend another Item of the same type that is 'visible' from its scope. In other words, an Item Y can extend a base Item X, as long as they are the same type, and one of the following visibility conditions holds:

1. X is a direct child of the Benchmark.
2. X is a direct child of a Group which is also an ancestor of Y.

For example, in the tiny Benchmark structure shown in Figure 3, it would be legal for Rule (g) to extend Rule (f) or extend Rule (h). It would not be legal for Rule (i) to extend Rule (m), because (m) is not visible from the scope of (i). It would not be legal for Rule (l) to extend Value (k), because they are not of the same type.

The ability for a Rule or Value to be extended by another gives Benchmark authors the ability to create variations or specialized versions of Items without making copies.

3.2. Object Content Details

The tables below show the properties that make up each data type in the XCCDF data model. Note that the properties that comprise a Benchmark or Item are an ordered sequence of property values, and the order in which they appear determines the order in which they are processed.

Properties with a data type of "HTML-enabled text" are string data that can include embedded formatting directives and hypertext links. Properties of type "string" may not include formatting. Properties of type "identifier" must be strings without spaces or formatting, obeying the definition of "NCName" from the XML Schema specification [2].

Note that, in this table, and in the similar tables throughout the section, a minimum value of 0 in the Count column indicates that the property is optional, and a minimum value of 1 or greater indicates that the property is mandatory.

Benchmark

Property	Type	Count	Description
id	identifier	1	Benchmark identifier, mandatory
status	string+date	1-n	Status of the Benchmark (see below) and date at which it attained that status (at least one status property must appear; if several appear, then the one with the latest date applies)
dc-status	*special*	0-1	Holds additional status information using the Dublin Core format
title	text	0-n	Title of the XCCDF Benchmark document

Property	Type	Count	Description
description	HTML-enabled text	0-n	Text that describes the Benchmark
version	string+date +URI	1	Version number of the Benchmark, with the date and time when the version was completed and an optional update URI
notice	HTML-enabled text	0-n	Legal notices or copyright statements about this Benchmark; each notice has a unique identifier and text value
front-matter	HTML-enabled text	0-n	Text for the front of the Benchmark document
rear-matter	HTML-enabled text	0-n	Text for the back of the Benchmark document
reference	*special*	0-n	A bibliographic reference for the Benchmark document: metadata or a simple string, plus an optional URL
platform-specification	*special*	0-1	A list of complex platform definition, in Common Platform Enumeration (CPE 2.3) language format [17]
platform	string	0-n	Target platforms for this Benchmark, each a URI referring to a platform listed in the community CPE 2.3 dictionary or an identifier defined in the CPE 2.3 Language platform-specification property
plain-text	string+ identifier	0-n	Reusable text blocks, each with a unique identifier; these can be included in other text blocks in the Benchmark
model	URI+ parameters	0-n	Suggested scoring model or models to be used when computing a compliance score for this Benchmark
profiles	Profile	0-n	Profiles that reference and customize sets of Items in the Benchmark
values	Value	0-n	Tailoring values that support Rules and descriptions in the Benchmark
groups	Group	0-n	Groups that comprise the Benchmark; each group may contain additional Values, Groups, and Rules
rules	Rule	0-n	Rules that comprise the Benchmark
test-results	TestResult	0-n	Benchmark test result records (one per Benchmark run)
metadata	*special*	0-n	Discovery metadata for the Benchmark
resolved	boolean	0-1	True if Benchmark has already undergone the resolution process (see Section 3.3)

Property	Type	Count	Description
style	string	0-1	Name of a benchmark authoring style or set of conventions to which this Benchmark conforms.
style-href	URI	0-1	URL of a supplementary stylesheet or schema extension that can be used to check conformance to the named style.
signature	*special*	0-1	A digital signature asserting authorship and allowing verification of the integrity of the Benchmark

Conceptually, a Benchmark contains Group, Rule, and Value objects, and it may also contain Profile and TestResult objects. For ease of reading and simplicity of scoping, all Profiles must precede all Groups, Rules, and Values. Groups can contain Values, Rules, and other Groups. Within any level of the Group hierarchy (including at the top level, within the Benchmark itself), Values must precede sibling Groups and Rules. All Values, Groups, and Rules must precede all TestResults. These objects may be directly embedded in the Benchmark, or incorporated via W3C standard XML Inclusion [11].

Each status property consists of a status string and a date. Permissible string values are "accepted", "draft", "interim", "incomplete", and "deprecated". Benchmark authors should mark their Benchmarks with a status to indicate a level of maturity or consensus. A Benchmark may contain one or more status properties, each holding a different status value and the date on which the Benchmark reached that status. In addition to the status field, an optional dc-status field is available. This field can contain additional status information using the Dublin Core schema.

The description field, as well as front-matter, back-matter, and notice fields, all can contain HTML markup. This allows authors to specify not only the content of these fields but how it would be displayed to a user. However, while this does allow authors to specify every detail of a field's display, authors may wish to utilize classifiers in these fields, using class attributes in or elements. These can tell tools the type of content contained within a text block and then allow the tool to display this content according to its own conventions. Authors may wish to do this so their content fits better with other automated formatting choices made by a tool or a stylesheet, and also because some tools may not support the complete range of HTML tags in their displays, thus causing some explicit formatting to be ignored. A list of recommended class values appears below. Authors are not limited to this list and tools are not required to have special formatting for any of the listed classifiers. However, both authors and vendors are encouraged to incorporate these class designators to provide more efficient display of content.

Class Value	Meaning
license	Indicates licensing and use information
copyright	Indicates copyright and ownership information
tangent	Indicates a block of text that contains tangentially related information (possibly appropriate for inclusion as a sidebar or a pop-up)
warning	Indicates pitfalls or cautions relative to the surrounding text. High-level and general warnings should appear in designated warning fields, if available.
critical	Indicates content of critical importance to the user

example	Indicates an example of some kind
instructions	Indicates special instructions to the user
default	General information. Empty or absent class attributes also imply "default" appearance. This tag allows authors to explicitly indicate text should appear in the default format.

Use of these class attribute values is applicable wherever HTML content can be included in the XCCDF document, including within the appropriate fields of Rules, Values, Groups, and Profiles.

Generally, XCCDF items can be qualified by platform using Common Platform Enumeration (CPE) Names, as defined in the CPE 2.3 Specification [17]. In CPE, a specific platform is identified by a unique name. Each Rule, Group, Profile, and the Benchmark itself may possess platform properties, each containing a CPE Name indicating the hardware or software platform to which the object applies. CPE 2.3 Names can express only unitary or simple platforms (e.g. "cpe:/o:microsoft:windows-nt:xp::pro" for Microsoft Windows XP Professional Edition). Sometimes, XCCDF rules require more complex qualification. The platform-specification property contains a list of one or more complex platform definitions expressed using CPE Language schema. Each definition bears a locally unique identifier. These identifiers may be used in platform properties in place of CPE Names.

Note that CPE Names may be used in a Benchmark or other objects without defining them explicitly. CPE Names for common IT platforms are generally defined in the community dictionary and may be used directly. Authors can use the platform-specification property to define complex platforms and assign them local identifiers for use in the Benchmark.

The Benchmark platform-specification property and platform properties are optional. Authors should use them to identify the systems or products to which their Benchmarks apply.

The plain-text properties allow commonly used text to be defined once and then re-used in multiple text blocks in the Benchmark. Note that each plain-text must have a unique id, and that the ids of other Items and plain-text properties must not collide. This restriction permits easier implementation of document generation and reporting tools.

Benchmark metadata allows authorship, publisher, support, and other information to be embedded in a Benchmark. Metadata should comply with existing commercial or government metadata specifications to allow Benchmarks to be discovered and indexed. The XCCDF data model allows multiple metadata properties for a Benchmark. The primary metadata format, which should appear in all published Benchmarks, is the simple Dublin Core Elements specification, as documented in [14].

The style and style-href properties may be used to indicate that a benchmark conforms to a specific set of conventions or constraints. For example, NIST is designing a set of style conventions for XCCDF benchmarks as part of the SCAP initiatives. The style property holds the name of the style (e.g. "SCAP 1.0") and the style-href property holds a reference to a stylesheet or schema that tools can use to test conformance to the style.

Note that a digital signature, if any, applies only to the Object in which it appears, but after inclusion processing (note: it may be impractical to use inclusion and signatures together). Any digital signature format employed for XCCDF Benchmarks must be capable of identifying the signer, storing all information needed to verify the signature (usually, a certificate or certificate

chain), and detecting any change to the content of the Benchmark. XCCDF tools that support signatures at all must support the W3C XML-Signature standard enveloped signatures [10].

Legal notice text is handled specially, as discussed in Section 3.3.

Item (base)

Property	Type	Count	Description
id	identifier	1	Unique object identifier, mandatory
title	text	0-n	Title of the Item (for human readers)
description	HTML-enabled text	0-n	Text that describes the Item
warning	HTML-enabled text	0-n	A cautionary note or caveat about the Item
status	string+date	0-n	Status of the Item and date at which it attained that status, optional
dc-status	*special*	0-1	Holds additional status information using the Dublin Core format
version	string+date +URI	0-1	Version number of the Benchmark, with the date and time when the version was completed and an optional update URI
question	string	0-n	Interrogative text to present to the user during tailoring
hidden	boolean	0-1	If this Item should be excluded from any generated documents (default: false)
prohibitChanges	boolean	0-1	If tools should prohibit changes to this Item during tailoring (default: false)
cluster-id	identifier	0-1	An identifier to be used from a Profile to refer to multiple Groups and Rules, optional
reference	*special*	0-n	A reference to a document or resource where the user can learn more about the subject of this Item: content is Dublin Core metadata or a simple string, plus an optional URL
metadata	*special*	0-n	Metadata associated with this Item
signature	*special*	0-1	Digital signature over this Item, optional

Every Item may include one or more status properties. Each status property value represents a status that the Item has reached and the date at which it reached that status. Benchmark authors can use status elements to record the maturity or consensus level for Rules, Groups, and Values in the Benchmark. If an Item does not have an explicit status property value given, then its status is taken to be that of the Benchmark itself. In addition to the status field, an optional dc-status field is available. This field can contain additional status information using the Dublin Core schema. Neither the status nor dc-status property are inherited under extension.There are several Item properties that give the Benchmark author control over how Items may be tailored and presented in documents. The 'hidden' property simply prevents an Item from appearing in generated documents. For example, an author might set the hidden property on incomplete Items

in a draft Benchmark. The 'prohibitChanges' property advises tailoring tools that the Benchmark author does not wish to allow end users to change anything about the Item.

The 'cluster-id' property is optional, but it provides a means to identify related Value, Group and Rule items throughout the Benchmark. Cluster identifiers need not be unique: all the Items with the same cluster identifier belong to the same cluster. A selector in a Profile can refer to a cluster, thus making it easier for authors to create and maintain Profiles in a complex Benchmark. The cluster-id property is not inherited.

The metadata field allows additional information to be associated with individual Items. These can be used to denote sources, special information, or other details.

Group :: Item

Property	Type	Count	Description
requires	identifier	0-n	The id of another Group or Rule in the Benchmark that must be selected for this Group to be applied and scored properly
conflicts	identifier	0-n	The id of another Group or Rule in the Benchmark that must be unselected for this Group to be applied and scored properly
selected	boolean	1	If true, this Group is selected to be processed as part of the Benchmark when it is applied to a target system; an unselected Group is not processed, and none of its contents are processed either (i.e., all descendants of an unselected group are implicitly unselected). Default is true. Can be overridden by a Profile
rationale	HTML-enabled text	0-n	Descriptive text giving rationale or motivations for abiding by this Group
platform	string	0-n	Platforms to which this Group applies, CPE 2.3 Names or CPE 2.3 platform specification identifiers
abstract	boolean	0-1	If true, then this Group is abstract and exists only to be extended (default: false). ***The use of this field is now deprecated.***
extends	identifier	0-1	An id of a Group on which to base this Group. ***The use of this field is now deprecated.***
weight	float	0-1	The relative scoring weight of this Group, for computing a compliance score; can be overridden by a Profile
values	Value	0-n	Values that belong to this Group, optional
groups	Group	0-n	Sub-groups under this Group, optional
rules	Rule	0-n	Rules that belong to this Group, optional

Although Groups have an extends attribute, use of this attribute is now deprecated and should be avoided. Likewise, a Group should never be marked as abstract.

The requires and conflicts properties provide a means for Benchmark authors to express dependencies among Rules and Groups. Their exact meaning depends on what sort of processing the Benchmark is undergoing, but in general the following approach should be applied: if a Rule or Group is about to be processed, and any of the Rules or Groups identified in a requires property have a selected property value of false or any of the Items identified in a conflicts property have a selected property value of true, then processing for the Item should be skipped and its selected property should be set to false.

The platform property of a Group indicates that the Group contains platform-specific Items that apply to some set of (usually related) platforms. First, if a Group does not possess any platform properties, then it applies to the same set of platforms as its enclosing Group or the Benchmark. Second, for tools that perform compliance checking on a platform, any Group whose set of platform property values do not include the platform on which the compliance check is being performed should be treated as if their selected property were set to false. Third, the platforms to which a Group applies should be a subset of the platforms applicable for the enclosing Benchmark. Last, if no platform properties appear anywhere on a Group or its enclosing Group or Benchmark, then the Group nominally applies to all platforms.

The weight property denotes the importance of a Group relative to its sibling in the same Group or its siblings in the Benchmark (for a Group that is a child of the Benchmark). Under some scoring models, scoring is computed independently for each collection of sibling Groups and Rules, then normalized as part of the overall scoring process. For more information about scoring, see Section 3.3.

Rule :: Item

Property	Type	Count	Description
selected	boolean	1	If true, this Rule is selected to be checked as part of the Benchmark when the Benchmark is applied to a target system; an unselected rule is not checked and does not contribute to scoring. Default is true. Can be overridden by a Profile
abstract	boolean	0-1	If true, then this Rule is abstract and exists only to be extended (default: false)
extends	identifier	0-1	The id of a Rule on which to base this Rule (must match the id of another Rule)
multiple	boolean	0-1	Whether this rule should be multiply instantiated. If false, then Benchmark tools should avoid multiply instantiating this Rule. If true, this Rule should produce a result for every instance of an assessed target. The default is false.

Property	Type	Count	Description
multi-check	boolean	0-1	If true, this Rule should produce a result for every referenced component check. If false,the check should produce only a single result regardless of the number of component checks. The default is false.
role	string	0-1	Rule's role in scoring and reporting; one of the following: "full", "unscored", "unchecked". Default is "full". Can be overridden by a Profile. ***The use of this field is now deprecated.***
severity	string	0-1	Severity level code, to be used for metrics and tracking. One of the following: "unknown", "info", "low", "medium", "high". Default is "unknown". Can be overridden by a Profile
weight	float	0-1	The relative scoring weight of this Rule, for computing a compliance score. Default is 1.0. Can be overridden by a Profile
rationale	HTML-enabled text	0-n	Some descriptive text giving rationale or motivations for complying with this Rule
platform	string	0-n	Platforms to which this Rule applies, CPE 2.3 Names or CPE 2.3 platform-specification identifiers
requires	identifier	0-n	The id of another Group or Rule in the Benchmark that should be selected for this Rule to be applied and scored properly
conflicts	identifier	0-n	The id of another Group or Rule in the Benchmark that should be unselected for this Rule to be applied and scored properly
ident	string+URI	0-n	A long-term, globally meaningful name for this Rule. May be the name or identifier of a security configuration issue or vulnerability that the Rule remediates. Has an associated URI that denotes the organization or naming scheme which assigns the name.(see below)
impact-metric	string	0-1	The impact metric for this rule, expressed as a CVSS score. (see below). ***The use of this field is now deprecated.***

Property	Type	Count	Description
profile-note	HTML-enabled text + identifier	0-n	Descriptive text related to a particular Profile. This property allows a Benchmark author to describe special aspects of the Rule related to one or more Profiles. It has an id that can be specified as the 'note-tag' property of a Profile (see the Profile description, below)
fixtext	HTML-enabled text	0-n	Prose that describes how to fix the problem of non-compliance with this Rule; each fixtext property may be associated with one or more fix property values
fix	*special*	0-n	A command string, script, or other system modification statement that, if executed on the target system, can bring it into full, or at least better, compliance with this Rule
check	*special*	0-n	The definition of, or a reference to, the target system check needed to test compliance with this Rule. A check consists of three parts: the checking system specification on which it is based, a list of Value objects to export, and the content of the check itself. If a Rule has several check properties with the same selector attribute, each must employ a different checking system
complex-check	*special*	0-1	A complex check is a boolean expression of other checks. At most one complex-check may appear in a Rule (see below)

A Rule can be based on (extend) another Rule. This means that the extending Rule inherits all the properties of the extended or base Rule, some of which it may override with new values. For any property that is allowed to appear more than once, the extending Rule gets the sequence of property values from the extended Rule, plus any of its own values for that property. For any property that is allowed to appear at most once, the extending Rule gets its own value for the property if one appears, otherwise it gets the extended Rule's value of that property. A Rule for which the abstract property is true should not be included in any generated document and must not be checked in any compliance test. Abstract Rules are removed during resolution (see Section 3.3).

The 'multiple' property is applicable in cases where there are many instances of a target. For example, a Rule may provide a recommendation about the configuration of application user accounts, but an application may have many user accounts. Each account would be considered an instance of the broader assessment target of user accounts. By setting 'multiple' to true, the Rule's author is directing that separate instances of the target to which the Rule can apply should be tested separately and the results recorded separately. By setting 'multiple' to false, the author is directing that the test results of such instances be combined. If the checking system does not combine these results automatically, the results of each instance should be ANDed together to produce a single result using the truth table that appears in the description of the <complex-

check> element on page 63. If the processing tool cannot perform multiple instantiation, or if multiple instantiation of the Rule is not applicable for the target system, then processing tools may ignore this property. In particular, it should be noted that, as of OVAL 5.7, OVAL checks cannot produce separate results for individual instances of a target.

The multi-check property is applicable in cases where multiple checks are executed to determine compliance with a single Rule. This situation can arise when a Rule's check includes a check-content-ref element that does not include a name property. The default behavior of a nameless check-content-ref is to execute all checks in the referenced file and AND their results together into a single XCCDF rule-result using the truth table that appears in the description of the <complex-check> element on page 63. If, however, the multi-check property is set to "true" and a nameless check-content-ref is used, each check in the targeted file should be reported as a separate rule-result.The weight property denotes the importance of a rule relative to other Rules. For more information about scoring, see Section 3.3.

The platform properties of a Rule indicate the platforms to which the Rule applies. Each platform property asserts a single CPE 2.3 Name or a CPE 2.3 Language identifier. If a Rule does not possess any platform properties, then it applies to the same set of platforms as its enclosing Group or Benchmark. For tools that perform compliance checking on a platform, if a Rule's set of platform property values does not include the platform on which the compliance check is being performed, the Rule should be treated as if its selected property were set to false. Any platform property value that appears on a Rule should be a member of the set of platform property values of the enclosing Benchmark. Finally, if no platform properties appear anywhere on a Rule or its enclosing Group or Benchmark, then the Rule applies to all platforms.

Each ident property contains a globally meaningful name in some security domain; the string value of the property is the name, and a Uniform Resource Identifier (URI) designates the scheme or organization that assigned the name. By setting an 'ident' property on a Rule, the Benchmark author effectively declares that the Rule instantiates, implements, or remediates the issue for which the name was assigned. For example, the ident value might be a Common Vulnerabilities and Exposures (CVE) identifier; the Rule would be a check that the target platform was not subject to the vulnerability named by the CVE identifier, and the URI would be that of the CVE Web site.

The check property consists of the following: a selector for use with Profiles, a field to indicate whether the result should be negated, a URI that designates the checking system or engine, a set of export declarations, and the check content. The checking system URI tells a compliance checking tool what processing engine it must use to interpret or execute the check. The nominal or expected checking system is MITRE's OVAL system (designated by http://oval.mitre.org/), but the XCCDF data model allows for alternative or additional checking systems. XCCDF also supports conveyance of tailoring values from the XCCDF processing environment down to the checking system, via export declarations. Each export declaration maps an XCCDF Value object id to an external name or id for use by the checking system. The check content is an expression or document in the language of the checking system; it may appear inside the XCCDF document (an *enveloped* check) or it may appear as a reference (a *detached* check).

In place of a 'check' property, XCCDF allows a 'complex-check' property. A complex check is a boolean expression whose individual terms are checks or complex-checks. This allows Benchmark authors to re-use checks in more flexible ways, and to mix checks written with different checking systems. A Rule may have at most one 'complex-check' property; on

inheritance, the extending Rule's complex-check replaces the extended Rule's complex-check. If both check properties and a complex-check property appear in a Rule, then the check properties must be ignored. The following operators are allowed for combining the constituents of a complex-check:

AND – if and only if all terms evaluate to Pass (true), then the complex-check evaluates to Pass.

OR – if any term evaluates to Pass, then the complex-check evaluates to Pass.

Truth-tables for the operators appear under their detailed descriptions in the next section. Note that each complex-check may also specify that the expression should be negated (boolean **not**).

The properties fixtext and fix exist to allow a Benchmark author to specify a way to remediate non-compliance with a Rule. The 'fixtext' property provides a prose description of the fix that needs to be made; in some cases this may be all that is possible to do in the Benchmark (e.g., if the fix requires manipulation of a GUI or installation of additional software). The 'fix' property provides a direct means of changing the system configuration to accomplish the necessary change (e.g., a sequence of command-line commands; a set of lines in a system scripting language like Bourne shell or in a system configuration language like Windows INF format; a list of update or patch ID numbers).

The fix and fixtext properties help tools support sophisticated facilities for automated and interactive remediation of Benchmark findings. The following attributes can be associated with a fix or fixtext property value:

- **strategy** – a keyword that denotes the method or approach for fixing the problem. This applies to both fix and fixtext. Permitted values: unknown (default), configure, combination, disable, enable, patch, policy, restrict, update.
- **disruption** – an estimate for how much disruption the application of this fix will impose on the target. This applies to fix and fixtext. Permitted values: unknown, low, medium, high.
- **reboot** – whether or not remediation will require a reboot or hard reset of the target. This applies to fix and fixtext. Permitted values: true (1) and false (0).
- **system** – a URI representing the scheme, language, engine, or process for which the fix contents are written. Appendix C defines several general-purpose URNs for this, but it is expected that tool vendors and system providers may need to define target-specific ones. This applies to fix only.
- **id/fixref** – these attributes will allow fixtext properties to be associated with specific fix properties (pair up explanatory text with specific fix procedures).
- **platform** – in case different fix scripts or procedures are required for different target platform types (e.g., different patches for Windows 2000 and Windows XP), this attribute allows a CPE 2.3 Name or CPE 2.3 Language definition to be associated with a fix property.

For more information, consult the definitions of the fix and fixtext elements in Section 4.2.

Value :: Item

Property	Type	Count	Description
value/ complex-value	string + id/*special*	1-n	The current value of this Value
default/ complex-default	string + id/*special*	0-n	Default value of this Value object, optional
type	string	0-1	The data type of the Value: "string", "number", or "boolean" (default: "string")
abstract	boolean	0-1	If true, then this Value is abstract and exists only to be extended (default: false)
extends	identifier	0-1	The id of a Value on which to base this Value
operator	string	0-1	The operator to be used for comparing this Value to some part of the test system's configuration (see list below)
lower-bound	number + identifier	0-n	Minimum legal value for this Value (applies only if type is 'number')
upper-bound	number + identifier	0-n	Maximum legal value for this Value (applies only if type is 'number')
choices	list + id	0-n	A list of legal or suggested values for this Value object, to be used during tailoring and document generation
match	string (regular expr.)	0-n	A regular expression which the Value must match to be legal (for more information, see [9])
interactive	boolean	0-1	Tailoring for this Value should also be performed during Benchmark application, optional (default is false)
interfaceHint	string	0-1	User interface recommendation for tailoring
source	URI	0-n	URI indicating where the Benchmark tool may acquire a value for this Value object

A Value is content that can be substituted into properties of other Items, including the interior of structured check specifications and fix scripts. New in XCCDF 1.2, Values may encapsulate values that are lists of simple types and/or which use externally defined datatypes. These new structures are supported by the complex-value, complex-default, and, within the choices field, the complex-choice elements. A single Value may use a mixture of both simple (i.e. a single number, string, or boolean value) and complex types. Apart from its ability to encapsulate different types of information, a complex field is used in the same way as its simple counterpart.

A tool may choose any convenient form to store a Value's value property, but the data type conveys how the value should be treated during Benchmark compliance testing. The data type property may also be used to give additional guidance to the user or to validate the user's input.

For example, if a Value object's type property was "number", then a tool might choose to reject user tailoring input that was not composed of digits. In the case of a list of values, the type attribute, if present, is applied to all elements of the list individually. The type property is ignored if the value contains externally defined XML structures. The default property holds a default value for the value property; tailoring tools may present the default value to users as a suggestion.A Value object may extend another Value object. In such cases, the extending object receives all the properties of the extended object, and may override them where needed. A Value object with the abstract property true should never be included in any generated document, and may not be exported to any compliance checking engine.

When defining a Value object, the Benchmark author may specify the operator to be used for checking compliance with the value. For example, one part of an operating system (OS) Benchmark might be checking that the configuration included a minimum password length; the Value object that holds the tailorable minimum could have type "number" and operator "greater than". Exactly how Values are used in rules may depend on the capabilities of the checking system. Tailoring tools and document generation tools may ignore the 'operator' property; therefore, Benchmark authors should included sufficient information in the description and question properties to make the role of the Value clear. The table below describes the operators permitted for each Value type.

Value Type	Available Operators	Remarks
number	equals, not equal, less than, greater than, less than or equal, greater than or equal	Default operator: equals
boolean	equals, not equal	Default operator: equals
string	equals, not equal, pattern match (pattern match means regular expression match; should comply with [9])	Default operator: equals
external-type	equals, not equal, less than, greater than, less than or equal, greater than or equal, pattern match (How these operators are interpreted may vary based on the data types. Not all operators may be appropriate for all possible content.)	Default operator: equals
lists	as component data type	Default operator: equals

A Value object includes several properties that constrain or limit the values that the Value may be given: value and/or complex-value, default and/or complex-default, match, choices, upper-bound, and lower-bound. Benchmark authors can use these Value properties to assist users in tailoring the Benchmark. These properties may appear more than once in a Value, and may be marked with a selector tag id. At most one instance of each may omit its selector tag. For more information about selector tags, see the description of the Profile object below.

The upper-bound and lower-bound properties constrain the choices for Value items or lists of items with a type property of 'number'. For any other type, they are meaningless. The bounds they indicate are always inclusive. For example, if the lower-bound property for a Value is given as "3", then 3 is a legal value.

The 'choices' property holds a list of one or more particular values for the Value object; the 'choices' property also bears a boolean flag, 'mustMatch', which indicates that the enumerated

choices are the only legal ones (mustMatch="1") or that they are merely suggestions (mustMatch="0"). The choices property should be used when there are a moderate number of known values that are most appropriate. For example, if the Value were the authentication mode for a server, the choices might be "password" and "pki".

The match property provides a regular expression pattern that a tool may apply, during tailoring, to validate user input. The 'match' property applies only when the Value type is 'string' or 'number' or lists thereof. For example, if the Value type was 'string', but the value was meant to be a Cisco IOS router interface name, then the Value match property might be set to "[A-Za-z]+ *[0-9]+(/[0-9.]+)*". This would allow a tailoring tool to reject an invalid user input like "f8xq+" but accept a legal one like "Ethernet1/3".

If a Value's prohibitChanges property is set to true, then it means that the Value's value may not be changed by the user. This might be used by Benchmark authors in defining values that are integral to compliance, such as a timeout value, or it might be used by enterprise security officers in constraining a Benchmark to more tightly reflect organizational or site security policies. (In the latter case, a security officer could use the extension facility to make an untailorable version of a Value object, without rewriting it.) A Value object can have a 'hidden' property; if the hidden property is true, then the Value should not appear in a generated document, but its value may still be used.

If the interactive property is set, it is a hint to the Benchmark checking tool to ask the user for a new value for the Value at the beginning of each application of the Benchmark. The checking tool is free to ignore the property if asking the user is not feasible or not supported. Similarly, the 'interfaceHint' property allows the Benchmark author to supply a hint to a benchmarking or tailoring tool about how the user might select or adjust the Value. The following strings are valid for the 'interfaceHint' property: "choice", "textline", "text", "date", and "datetime".

The source property allows a Benchmark author to supply a URI, possibly tool-specific, that indicates where a benchmarking or tailoring tool may acquire values, value bounds, or value choices.

Profile

Property	Type	Count	Description
id	identifier	1	Unique identifier for this Profile
title	string	1-n	Title of the Item, for human readers
description	HTML-enabled text	0-n	Text that describes the Profile
extends	identifier	0-1	The id of a Profile on which to base this Profile
abstract	boolean	0-1	If true, then this Profile exists solely to be extended by other Profiles, and may not be applied to a Benchmark directly; optional (default: false)
note-tag	identifier	0-1	Tag identifier to match profile-note properties in Rules
status	string + date	0-n	Status of the Profile and date at which it attained that status

Property	Type	Count	Description
dc-status	*special*	0-1	Holds additional status information using the Dublin Core format
version	string + date	0-1	Version of the Profile, with timestamp and update URI
prohibitChanges	boolean	0-1	Whether or not tools should prohibit changes to this Profile (default: false)
platform	string	0-n	A target platform for this Profile, a CPE 2.3 Name or platform-specification identifier. Multiple platform names may be listed if the Profile applies to several platforms
reference	string + URL	0-n	A reference to a document or resource where the user can learn more about the subject of this Profile: a string and optional URL
selectors	*special*	0-n	References to Groups, Rules, and Values, see below (references may be the unique id of an Item, or a cluster id)
metadata	*special*	0-n	Metadata associated with this Profile
signature	*special*	0-1	Digital signature over this Profile, optional

A Profile object is a named tailoring of a Benchmark. While a Benchmark can be tailored in place, by setting properties of various objects, only Profiles allow one Benchmark document to hold several independent tailorings.

A Profile can extend another Profile in the same Benchmark. The set of platform and reference selector properties of the extended Profile appear before the corresponding properties in the extending Profile. Selector properties of the extended Profile also appear before the selector properties of the extending Profile. However, when selectors have overlapping idref attributes the selector from the extended Profile is not copied – effectively resulting in its replacement by the corresponding selector in the extending Profile. Inheritance of title, description, and reference properties are handled in the same way as for Rule objects.

The note-tag property is a simple identifier. It specifies which profile-note properties on Rules should be associated with this Profile.

Benchmark authors can use the Profile's 'status' property to record the maturity or consensus level of a Profile. If the status is not given explicitly in a Profile definition, then the Profile is taken to have the same status as its parent Benchmark. In addition to the status field, an optional dc-status field is available. This field can contain additional status information using the Dublin Core schema. Neither the status nor dc-status property is inherited.

Each Profile contains a list of selectors which express a particular customization or tailoring of the Benchmark. There are five kinds of selectors:

- **select** - a Rule/Group selector. This selector designates a Rule, Group, or cluster of Rules and Groups. It overrides the selected property on the designated Items. It provides a means for including or excluding rules from the Profile.

- **set-value** – a Value selector. This selector overrides the value property of a Value object, without changing any of its other properties. It provides a means for directly specifying the value of a variable to be used in compliance checking or other Benchmark processing. This selector may also be applied to the Value items in a cluster, in which case it overrides the value properties of all of them.
- **set-complex-value** – similar to the set-value selector, this selector supports the direct specification of complex value types such as lists or externally-defined types. Like set-value, set-complex-value may also be applied to Value items in a cluster.
- **refine-rule** – a Rule/Group selector. This selector allows the Profile author to select check statements, override the scoring weight, severity, and role of a Rule, Group, or cluster of Rules and Groups. Despite the name, this selector does apply for Groups, but only to their weight property.
- **refine-value** – a Value selector. This selector designates the Value constraints to be applied during tailoring, for a Value object or the Value members of a cluster. It provides a means for authors to impose different constraints on tailoring for different profiles. (Constraints must be designated with a selector id. For example, a particular numeric Value might have several different sets of 'value', 'upper-bound', and 'lower-bound' properties, designated with different selector ids. The refine-value selector tells benchmarking tools which value to employ and bounds to enforce when that particular profile is in effect.)

All of the selectors except set-value and set-complex-value can include remark elements, to allow the benchmark author to add explanatory material to individual elements of the Profile.

Selectors are applied in the order they appear within the Profile. For selectors that refer to the same Item or cluster, this means that later selectors can override or change the actions of earlier ones. See page 43 for more details about selector processing.

TestResult

Property	Type	Count	Description
id	identifier	1	Identifier for this TestResults object
benchmark	URI+id	0-1	Reference to Benchmark; mandatory if this TestResults object is in a file by itself, optional otherwise
version	string	0-1	The version number string copied from the Benchmark
title	string	0-n	Title of the test, for human readers
remark	string	0-n	A remark about the test, possibly supplied by the person administering the Benchmark checking run, optional
organization	string	0-n	The name of the organization or enterprise responsible for applying this Benchmark and generating this result

Property	Type	Count	Description
identity	string+boolean	0-1	Information about the system identity employed during application of the Benchmark
start-time	timestamp	0-1	Time when test began
end-time	timestamp	1	Time when test was completed and the results recorded
test-system	string	0-1	Name of the test tool or program that generated this TestResult object; should be a CPE 2.3 Name [17]
target	string	1-n	Name of the target system whose test results are recorded in this object
target-address	string	0-n	Network address of the target
target-facts	*special*	0-1	A sequence of named facts about the target system or platform, including a type qualifier
platform	string	0-n	CPE 2.3 platform names indicating platforms which the target system was found to meet. Tools may insert multiple platform names if the target system met multiple relevant platform definitions
profile	identifier	0-1	The identifier of the Benchmark profile used for the test, if any
set-value	string + id	0-n	Specific settings for Value objects used during the test, one for each Value
set-complex-value	*special*	0-n	Specify settings for Value objects used during the test when the given value is set to a complex type, such as a list or an externally-defined type.
rule-results	*special*	1-n	Outcomes of individual Rule tests, one per Rule instance
score	float + URI	1-n	An overall score for this Benchmark test; at least one must appear
metadata	*special*	0-n	Metadata associated with this TestResult
signature	*special*	0-1	Digital signature over this TestResult object

A TestResult object represents the results of a single application of the Benchmark to a single target platform. The properties of a TestResult object include test time, the identity and other facts about the system undergoing the test, and Benchmark information. If the test was conducted using a specific Profile of the Benchmark, then a reference to the Profile may be included. Also, multiple set-value and set-complex-value properties may be included, giving the identifier and value for the Values that were used in the test. The 'test-system' property gives the

CPE 2.3 Name for the testing tool or application responsible for generating this TestResult object.

At least one target property must appear in the TestResult object. Each appearance of the property supplies a name by which the target host or device was identified at the time the test was run. The name may be any string, but applications should include the fully qualified Domain Name System (DNS) name whenever possible. The 'target-address' property is optional; each appearance of the property supplies an address which was bound by the target at the time the test was run. Typical forms for the address include: Internet Protocol version 4 (IPv4) address, Internet Protocol version 6 (IPv6) address, and Ethernet media access control (MAC) address.

The 'organization' property documents the organization, enterprise, or group responsible for the benchmark. The property may appear multiple times, to indicate multiple levels of an organizational hierarchy, in which case the highest-level organization should appear first, followed by subordinate organizations.

The identity property provides up to three pieces of information about the system identity used to apply the benchmark and generate the findings encapsulated by this TestResult object. The three pieces of information are:

- **authenticated** – whether the identity was authenticated with the target system during the application of the benchmark [boolean].
- **privileged** – whether the identity was granted privileges beyond those of a normal system user, such as superuser on Unix or LocalSystem rights on Windows [boolean].
- **name** – the name of the authenticated identity [string]. (The names of privileged identities are considered sensitive for most systems. Therefore, this part of the identity property may be omitted.)

The target-facts list is an optional part of the TestResult object. It contains a list of zero or more individual facts about the target system or platform. Each fact consists of the following: a name (URI), a type ("string", "number", or "boolean"), and the value of the fact itself.

The metadata field allows additional information to be associated with TestResults. For example, this field can hold a copy of the metadata field of the Benchmark which served as the source for these results. This is especially useful if the TestResult will be separated from its source Benchmark as the publication and support information in the Benchmark can travel with the TestResults. Tools may also add their own metadata to the TestResults they produce.

The main content of a TestResult object is a collection of rule-result records, each giving the result of a single instance of a rule application against the target. The TestResult must include one rule-result record for each Rule that was selected in the resolved Benchmark; it may also include rule-result records for Rules that were unselected in the Benchmark. A rule-result record contains the properties listed below. For more information about applying and scoring Benchmarks, see page 49.

The XCCDF schema allows a TestResult to exist alone in its own file. In addition, tools could construct their own file formats for storage of XCCDF TestResults. While this can be an efficient way to store results, users and tool developers should include mechanisms so that TestResults can retain their context even if separated from their source Benchmark since the TestResult only has meaning relative to the Rules that produced it. There are several ways to preserve this context even if TestResults are separated from their source Benchmarks. These can include

making sure that the benchmark element in the TestResult is a persistent link to the specific document and version of the Benchmark that produced the result, filling in the relevant version, organization, and similar fields, and/or through the use of descriptive metadata. This document does not mandate any specific approach, but tools and users should ensure some mechanism is in place to avoid context loss.

TestResult/rule-result

Property	Type	Count	Description
idref	identifier	1	Identifier of a Benchmark Rule (from the Benchmark designated in the TestResult)
time	timestamp	0-1	Time when application of this instance of this Rule was completed
version	string	0-1	The version number string copied from the version property of the Rule
severity	string	0-1	The severity string code copied from the Rule; defaults to “unknown”
role	string	0-1	The role string copied from the role property of the Rule. ***The use of this field is now deprecated.***
weight	float	0-1	The weight number copied from the weight property of the Rule
ident	string + URI	0-n	A globally meaningful name and URI for the issue or vulnerability, copied from the Rule
result	string	1	Result of this test: one of status values listed below
override	*special*	0-n	An XML block explaining how and why an auditor chose to override the Rule’s result status
instance	string	0-n	Name of the target system component to which this result applies, for multiply instantiated Rules. May also include context and hierarchy information for nested contexts (see below for details)
metadata	*special*	0-n	Metadata associated with this rule-result
message	string + code	0-n	Diagnostic messages from the checking engine, with optional severity (this would normally appear only for result values of “fail” or “error”)
fix	string	0-1	Fix script for this target platform, if available (would normally appear only for result values of “fail”)

Property	Type	Count	Description
check	*special*	0-n	Encapsulated or referenced results to detailed testing output from the checking engine (if any); if multiple checks were executed as part of a complex-check, then data for each may appear here

The result of a single test may be one of the following:

pass – the target system or system component satisfied all the conditions of the Rule; a pass result contributes to the weighted score and maximum possible score. [Abbreviation: P]

fail – the target system or system component did not satisfy all the conditions of the Rule; a fail result contributes to the maximum possible score. [Abbreviation: F]

error – the checking engine encountered a system error and could not complete the test, therefore the status of the target's compliance with the Rule is not certain. This could happen, for example, if a Benchmark testing tool were run with insufficient privileges. [Abbreviation: E]

unknown – the testing tool encountered some problem and the result is unknown. For example, a result of 'unknown' might be given if the Benchmark testing tool were unable to interpret the output of the checking engine. [Abbreviation: U]

notapplicable – the Rule was not applicable to the target of the test. For example, the Rule might have been specific to a different version of the target OS, or it might have been a test against a platform feature that was not installed. Results with this status do not contribute to the Benchmark score. [Abbreviation: N]

notchecked – the Rule was not evaluated by the checking engine. This status is designed for Rules that have no check properties. It may also correspond to a status returned by a checking engine. Results with this status do not contribute to the Benchmark score. [Abbreviation: K]

notselected – the Rule was not selected in the Benchmark. Results with this status do not contribute to the Benchmark score. [Abbreviation: S]

informational – the Rule was checked, but the output from the checking engine is simply information for auditor or administrator; it is not a compliance category. This status value is designed for Rules whose main purpose is to extract information from the target rather than test compliance. Results with this status do not contribute to the Benchmark score. [Abbreviation: I]

fixed – the Rule had failed, but was then fixed (possibly by a tool that can automatically apply remediation, or possibly by the human auditor). Results with this status should be scored the same as **pass**. [Abbreviation: X]

The instance property specifies the name of a target subsystem or component that passed or failed a Rule. This is important for Rules that apply to components of the target system, especially when a target might have several such components. For example, a Rule might specify a particular setting that needs to be applied on every interface of a firewall; for Benchmark compliance results, a firewall target with three interfaces would have three rule-result elements with the same rule id, each with an independent value for the 'result' property. For more discussion of multiply instantiated Rules, see page 49.

The metadata field allows additional information to be associated with individual rule-results. For example, this field could hold a copy of the metadata field of the Rule which served as the source for these results. Tools may also add their own metadata to the rule-results they produce

The 'check' property consists of the URI that designates the checking system, and detailed output data from the checking engine. The detailed output data can take the form of encapsulated XML or text data, or it can be a reference to an external URI. (Note: this is analogous to the form of the Rule object's check property, used for referring to checking engine input.)

The override property provides a mechanism for an auditor to change the Rule result assigned by the Benchmark checking tool. This is necessary (a) when checking a rule requires reviewing manual procedures or other non-IT conditions, and (b) when a Benchmark check gives an inaccurate result on a particular target system. The override element contains the following properties:

Property	Type	Count	Description
time	timestamp	1	When the override was applied
authority	string	1	Name or other identification for the human principal authorizing the override
old-result	string	1	The rule result status before this override
new-result	string	1	The new, override rule result status
remark	string	1	Rationale or explanation text for why or how the override was applied

XCCDF is not intended to be a database format for detailed results; the TestResult object offers a way to store the results of individual tests in modest detail, with the ability to reference lower-level testing data.

3.3. Processing Models

The XCCDF specification is designed to support automated XCCDF document processing by a variety of tools. There are five basic types of processing that a tool might apply to an XCCDF document:

1. **Tailoring.** This type of processing involves loading an XCCDF document, allowing a user to set the value property of Value items and the selected property of all Items, and then generating a tailored XCCDF output document.
2. **Document Generation.** This type of processing involves loading an XCCDF document and generating textual or formatted output, usually in a form suitable for printing or human perusal.

3. **Transformation.** This is the most open-ended of the processing types: it involves transforming an XCCDF document into a document in some other representation. Typically, a transformation process will involve some kind of stylesheet or specification that directs the transformation (e.g., an Extensible Stylesheet Language Transformation [XSLT] stylesheet). This kind of processing can be used in a variety of contexts, including document generation.
4. **Compliance Checking.** This is the primary form of processing for XCCDF documents. It involves loading an XCCDF document, checking target systems or data sets that represent the target systems, computing one or more scores, and generating one or more XCCDF TestResult objects. Some tools might also generate other outputs or store compliance information in some kind of database.
5. **Test Report Generation.** This form of processing can be performed only on an XCCDF document that includes one or more TestResult objects. It involves loading the document, traversing the list of TestResult objects, and generating non-XCCDF output and/or human-readable reports about selected ones.

Tailoring, document generation, and compliance checking all share a similar processing model consisting of two steps: loading and traversal. The processing sequence required for loading is described in the subsection below. Note that loading must be complete before traversal begins. When loading is complete, a Benchmark is said to be *resolved*.

Loading Processing Sequence

Before any loading begins, a tool should initialize an empty set of legal notices and an empty dictionary of object ids.

Sub-Step	Description
Loading.Import	Import the XCCDF document into the program and build an initial internal representation of the Benchmark object, Groups, Rules, and other objects. If the file cannot be read or parsed, then Loading fails. (At the beginning of this step, any inclusion processing specified with XInclude elements should be performed. The resulting XML information set should be validated against the XCCDF schema given in Appendix A.) Go to the next step: Loading.Noticing.
Loading.Noticing	For each notice property of the Benchmark object, add the notice to the tool's set of legal notices. If a notice with an identical id value is already a member of the set, then an error should be raised. If the Benchmark's resolved property is set, then Loading succeeds, otherwise go to the next step: Loading.Resolve.Items.
Loading.Resolve.Items	For each Item in the Benchmark that has an extends property, resolve it by using the following steps:(1) resolve the extended Item, (2) insert the necessary property sequences from the extended Item into the appropriate locations in the extending Item, (3) remove all but the last instance of duplicate properties and apply property overrides, and (4) remove the extends property. If any Item's extends property identifier does not match the identifier of a visible Item of the same type, then Loading fails. If the directed graph formed by the extends

Sub-Step	Description
	properties includes a loop, then Loading fails. Otherwise, go to the next step: Loading.Resolve.Profiles.
Loading.Resolve.Profiles	For each Profile in the Benchmark that has an extends property, resolve the set of properties in the extending Profile by applying the following steps: (1) resolve the extended Profile, (2) insert the necessary property sequences from the extended Profile into the appropriate locations in the extending Profile, (3) remove all but the last instance of duplicate properties. If any Profile's extends property identifier does not match the identifier of another Profile in the Benchmark, then Loading fails. If the directed graph formed by the extends properties of Profiles includes a loop, then Loading fails. Otherwise, go to Loading.Resolve.Abstract.
Loading.Resolve.Abstract	For each Item in the Benchmark for which the abstract property is true, remove the Item. For each Profile in the Benchmark for which the abstract property is true, remove the Profile. Go to the next step: Loading.Resolve.Finalize.
Loading.Resolve.Finalize	Set the Benchmark resolved property to true; Loading succeeds.

If the Loading step succeeds for an XCCDF document, then the internal data model should be complete, and every Item should contain all of its own content. An XCCDF file that has no extends properties is called a resolved document. Only resolved XCCDF documents should be subjected to Transformation processing.

XML Inclusion processing must happen before any validation or processing. Typically, it will be performed by the XML parser as the XML file is processed at the beginning of Loading.Import. XML Inclusion processing is independent of all XCCDF processing.

During the Loading.Resolve.Items and Loading.Resolve.Profiles steps, the processor must flatten inheritance relationships. The conceptual model for XCCDF object properties is a list of name-value pairs; property values defined in an extending object are added to the list inherited from the extending object. Where they are added to this list depends on the inheretance processing model for the given property.

There are five different inheritance processing models for Item and Profile properties.

- **None** – the property value or values are not inherited.
- **Prepend** – the property values are inherited from the extended object, but values on the extending object come first, and inherited values follow.
- **Append** – the property values are inherited from the extended object; additional values may be defined on the extending object.
- **Replace** – the property value is inherited; a property value explicitly defined on the extending object replaces an inherited value.
- **Override** –if explicitly tagged as 'override', the property is processed as if it uses the Replace model. Otherwise, the property is processed as if it uses the Append model.

The table below shows the inheritance processing model for each of the properties supported on Rule, Value, and Profile objects.

Processing Model	Properties	Remarks
None	abstract, cluster-id, extends, id, signature, status, dc-status	These properties cannot be inherited at all; they must be given explicitly
Prepend	source, choices	
Append	requires, conflicts, ident, fix, value, complex-value, default, complex-default, lower-bound, upper-bound, match, select, refine-value, refine-rule, set-value, set-complex-value, profileNote	Additional rules may apply during Benchmark processing, tailoring, or report generation
Replace	hidden, prohibitChanges, selected, version, weight, operator, interfaceHint, check, complex-check, role, severity, type, interactive, multiple, note-tag, impact-metric	For the check property, checks with different systems or different selectors are considered different properties
Override	title, description, platform,question, rationale, warning, reference, fixtext	For properties that have a locale (xml:lang specified), values with different locales are considered to be different properties

The second step of processing is Traversal. The concept behind Traversal is basically a pre-order, depth-first walk through all the Items that make up a Benchmark. However, Traversal works slightly differently for each of the three kinds of processing, as described further below.

Benchmark Processing Algorithm

The id of a Profile may be specified as input for Benchmark processing. A single instance of document generation or assessment would involve the selection of, at most, a single Profile.

Sub-Step	Description
Benchmark.Front	Process the properties of the Benchmark object
Benchmark.Profile	If a Profile id was specified, then apply the settings in the Profile to the Items of the Benchmark. At most one Profile id can be specified.
Benchmark.Content	For each Item in the Benchmark object's items property, initiate Item.Process
Benchmark.Back	Perform any additional processing of the Benchmark object properties

The sub-steps Front and Back will be different for each kind of processing, and each tool may perform specialized handling of Benchmark properties. For document generation, Profiles may be processed separately as part of Benchmark.Back, to generate part of the output document.

Item Processing Algorithm

Sub-Step	Description
Item.Process	Check the contents of the requires and conflicts properties, and if any required Items are unselected or any conflicting Items are selected, then set the selected and allowChanges properties to false.
Item.Select	If any of the following conditions holds, cease processing of this Item. 1. The processing type is Tailoring, and the selected property is false. 2. The processing type is Document Generation, and the hidden property is true. 3. The processing type is Compliance Checking, and the selected property is false. 4. The processing type is Compliance Checking, and the current platform (if known by the tool) is not a member of the set of platforms for this Item.
Group.Front	If the Item is a Group, then process the properties of the Group.
Group.Content	If the Item is a Group, then for each Item in the Group's items property, initiate Item.Process.
Rule.Content	If the Item is a Rule, then process the properties of the Rule.
Value.Content	If the Item is a Value, then process the properties of the Value.

Processing the properties of an Item is the core of Benchmark processing. The list below describes some of the processing in more detail.

- For Tailoring, the key to processing is to query the user and incorporate the user's response into the data. For a Group or Rule, the user should be given a yes/no choice. For a Value item, the user should be given a chance to supply a string value, possibly validated using the type property. The output of a tailoring tool will usually be another XCCDF file.
- For Document Generation, the key to processing is to generate an output stream that can be formatted as a readable or printable document. The exact formatting discipline will depend on the tool and the target output format. In general, the selected property is not germane to Document Generation. A Rule, Group, or Value with the hidden property set to true should not appear in the generated document. The platform properties may be used during Document Generation for generation of platform-specific versions of a document.
- For Compliance Checking, the key to processing is applying the Rule checks to the target system or collecting data about the target system. Tools will vary in how they do this and in how they generate output reports. It is also possible that some Rule checks will need to be applied to multiple contexts or features of the target system, generating multiple pass or fail results for a single Rule object.

Note that it is possible (but inadvisable) for a Benchmark author to set up circular dependencies or conflicts using the requires and conflicts properties. To prevent ambiguity, tools must process the Items of the Benchmark in order, and must not change the selected property of any Rule or Group more than once during a processing session. It should be emphasized that Groups and

Rules will only change from selected to deselected due to their requires and conflicts properties. No item will ever be changed from deselected to selected due to these properties. Also, requires/conflicts statements will only change their containing Item – they do not modify other Items in the Benchmark. Finally, note also that requires/conflicts properties are only evaluated once. Later changes to a Benchmark's state might result in de-selections that would cause a previous evaluation of requires/conflicts properties to come to a different conclusion. However, prior evaluations may never change, even if their results become "incorrect" after subsequent Items are processed.

Requires and conflicts properties are capable of expressing complicated dependencies. A requires property can hold a space-separated list of references to Items. An individual requires property would evaluate to true if at least one of the referenced Items was selected at the time the property was evaluated. As such, a requires property that looked like <requires idref="A B C"> could be read as "requires that Item A or Item B or Item C be selected". A conflicts property may only reference a single Item so would only evaluate to true if the referenced item was not selected at the time the property was evaluated. Rules and Groups may contain any number of requires and conflicts properties and if any of these properties do not evaluate to true, then that Item becomes deselected. As such, one could say that the results of each individual requires and conflicts property is ANDed together to produce a final determination of whether a given Item's requires/conflicts parameters are met.

Requires/conflicts examples

This section provides a few examples of the processing of requires and conflicts properties. In all examples, it is assumed that application of Profiles and/or manual tailoring has already occurred.

Example #1 – Simple requires/conflicts example

Below is a simple example of a Rule that uses requires and conflicts properties:

Example 1 – Simple requires/conflicts Example

```
<Rule id="Rule1" selected="true">
   ..
   <requires idref="Rule2 Rule3"/>
   <requires idref="Group1" />
   <conflicts idref="Rule4" />
   ...
</Rule>
```

The above Rule would only be selected if at least one of Rule2 or Rule3 was selected, if Group1 was selected, and if Rule4 was not selected. Expressed in boolean logic format, Rule1's requires/conflicts parameters would be met if and only if:

```
((Rule2 OR Rule3) AND Group1 AND ~Rule4)
```

In the above algebra, we say that a name evaluates to "true" if and only if the named Item is selected. Note that if Rule1 were already de-selected, the requires/conflicts evaluation becomes moot – the Rule would never change to selected even if all its requires and conflicts properties were met. Likewise, Rule1's requires and conflicts statements never affect Rule2, Rule3, Rule4, or Group1.

Example #2 – Ordering of requires/conflicts

As mentioned earlier, an Item's compliance with its requires/conflicts properties is only evaluated at one point in time and subsequent changes to the Benchmark's state, even if they would make those evaluations "incorrect" if re-run, do not change the prior results. Consider the following Rules:

Example 2 – Ordered requires/conflicts Example

```
<Rule id="Rule1" selected="true">
    ...
    <requires idref="Rule2">
    ...
</Rule>
<Rule id="Rule2" selected="true">
    ...
    <requires idref="Rule3">
    ...
</Rule>
<Rule id="Rule3" selected="false">
    ...
    <requires idref="Rule4">
    ...
</Rule>
<Rule id="Rule4" selected="true">
    ...
</Rule>
```

In the above example, Rule1 requires Rule2, Rule2 requires Rule3, and Rule3 requires Rule4, although since Rule3 is already de-selected, its requires statements are irrelevant. Looking at the above scenario, one might be tempted to believe that Rule1, Rule2, and Rule3 will all end up de-selected, but this is not the case. The following steps show how Item Processing of these Rules would proceed. For this example, we will assume we are doing compliance checking.

1. **Item.Process(Rule1)** – Because Rule2 is required, we check to see if Rule2 is selected. It is, so we make no change to Rule1's selection status.
2. **Item.Select(Rule1)** – Rule1 is selected. Continue processing Rule1.
3. **Rule.Content(Rule1)** – Process Rule1's content.
4. **Item.Process(Rule2)** – Because Rule3 is required, we check to see if Rule3 is selected. It is not, so we set selected on Rule2 to false and allowChanges to false.
5. **Item.Select(Rule2)** – Rule2 is not selected. Terminate processing of Rule2.
6. **Item.Process(Rule3)** – Because Rule4 is required, we check to see if Rule3 is selected. It is, but Rule 3 is already de-selected and remains so.
7. **Item.Select(Rule3)** – Rule3 is not selected. Terminate processing of Rule3.
8. **Item.Process(Rule4)** – Rule4 has no requires/conflicts properties so this step is skipped.
9. **Item.Select(Rule4)** – Rule4 is selected. Continue processing Rule4.
10. **Rule.Content(Rule4)** – Process Rule4's content.

The final result was that Rule1 and Rule4 were selected and processed while Rule2 and Rule3 were de-selected and not processed. This happens even though Rule1 requires Rule2. Because we have completed processing of Rule1's content before we start processing of Rule2, by the time we realize that Rule2's requires statement cannot be met and Rule2 becomes de-selected, the effect this change would have on Rule1 is moot because Rule1 has already been run.

This example demonstrates the importance of processing Items in the order in which they appear in the Benchmark XML. If an interpreter processed these Items in a different order (for example, from the bottom up), this would result in a different set of Rule contents being processed, which would violate the XCCDF specification.

Example #3 – Requires, conflicts, and Groups

Example #2 shows how a Rule's content might be processed even though a Rule that it requires is not (eventually) selected. Another way this can happen is if a required Rule is contained in a de-selected Group. Consider the following example:

Example 3 – Using requires/conflicts With Groups

```
<Rule id="Rule1" selected="false">
    ...
</Rule>
<Group id="Group1" selected="false">
    ...
    <Rule id="Rule2" selected="true">
        ...
        <requires idref="Rule1" />
        ...
    </Rule>
    ...
</Group>
<Rule id="Rule3" selected="true">
    ...
    <requires idref="Rule2" />
    ...
</Rule>
```

Because Group1 is de-selected, none of its contents will ever be processed. Thus, even though Rule2 would never be run and even though its requires property would not be met, it remains selected and, as such, allows the requires statement of Rule3 to evaluate to true. The following steps show how Item Processing of these Rules would proceed. For this example, we will assume we are doing compliance checking.

1. **Item.Process(Rule1)** – Rule1 has no requires/conflicts properties so this step is skipped.
2. **Item.Select(Rule1)** – Rule1 is not selected. Terminate processing of Rule1.
3. **Item.Process(Group1)** – Group1 has no requires/conflicts properties so this step is skipped.
4. **Item.Select(Group1)** – Group1 is not selected. Terminate processing of Group1. *Note that we never get to the Group.Content step so Rule2 never undergoes any form of processing.*

5. **Item.Process(Rule3)** – Because Rule2 is required, we check to see if Rule2 is selected. It is, so we make no change to Rule1's selection status.
6. **Item.Select(Rule3)** – Rule3 is selected. Continue processing Rule3.
7. **Rule.Content(Rule3)** – Process Rule3's content.

We see that Rule3 is run even though it requires a Rule that is not run.

Profile Selector Processing

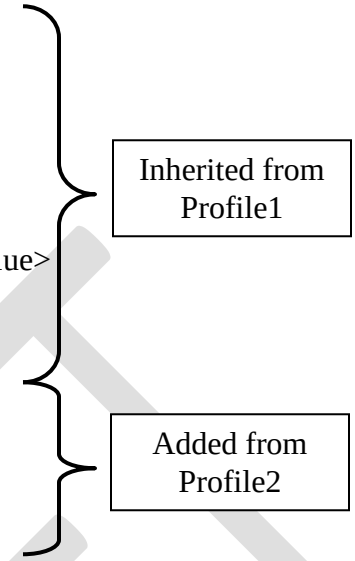
Profile selectors (select, refine-value, set-value, set-complex-value, and refine-rule) are processed in the order in which they appear in the XML. Given that these selectors are processed under the "append" extension processing model, this allows an extending Profile to override the selectors of the Profile it extends. Consider the following example of selector processing:

Example 4 – Profile Extension Example

```
<Profile id="Profile1" abstract="true">
   ...
   <select idref="Rule1" selected="true" />
   <select idref="Cluster1" selected="false" />
   <select idref="Group1" selected="true" />
   <refine-value idref="Value1" selector="sel1" />
   <refine-value idref="Cluster2" selector="sel2" />
   <set-value idref="Value2">NEWVALUE</set-value>
   <refine-rule idref="Rule2" selector="sel3" />
   <refine-rule idref="Cluster3" selector="sel1" />
</Profile>
<Profile id="Profile2" extends="Profile1">
   ...
   <select idref="Rule1" selected="false" />
   <select idref="Rule3" selected="true" />
   <refine-value idref="Cluster2" selector="sel5" />
   <refine-rule idref="Rule2" selector="sel6" />
</Profile>
```

Because Profile selectors are appended under extension, after Loading steps have completed, Profile2's effective list of selectors would look like the following (with numbers added for reference):

1. <select idref="Rule1" selected="true" />
2. <select idref="Cluster1" selected="false" />
3. <select idref="Group1" selected="true" />
4. <refine-value idref="Value1" selector="sel1" />
5. <refine-value idref="Cluster2" selector="sel2" />
6. <set-value idref="Value4">NEWVALUE</set-value>
7. <refine-rule idref="Rule2" selector="sel3" />
8. <refine-rule idref="Cluster3" selector="sel1" />
9. <select idref="Rule1" selected="false" />
10. <select idref="Rule3" selected="true" />
11. <refine-value idref="Cluster2" selector="sel5" />
12. <refine-rule idref="Rule5" selector="sel6" />



Further assume the existence and initial configuration of the following Rules, Groups, and Values:

Item	id	cluster-id	selected	Defined Selectors
Rule	Rule1	Cluster2	false	(empty)
Rule	Rule2		true	(empty), sel3
Rule	Rule3	Cluster1	true	(empty)
Rule	Rule4	Cluster1	true	sel1, sel5
Rule	Rule5	Cluster3	true	sel1
Group	Group1	Cluster1	true	
Group	Group2	Cluster1	true	
Value	Value1			(empty), sel1, sel2
Value	Value2	Cluster2		(empty), sel1, sel2, sel5
Value	Value3	Cluster2		(empty), sel1, sel5, sel6
Value	Value4	Cluster3		(empty), sel4

Before application of the profile, the state of the Benchmark is as follows:

Item id	Selected?	Applicable Selector
Rule1	not selected	(empty)
Rule2	selected	(empty)
Rule3	selected	(empty)
Rule4	selected	-none-
Rule5	selected	-none-

Group1	selected	-none-
Group2	selected	-none-
Value1		(empty)
Value2		(empty)
Value3		(empty)
Value4		(empty)

Assuming Profile2 is selected then the Benchmark.Profile processing step will cause the following changes to the resolved Benchmark as each selector is processed:

1. Rule1 becomes "selected"
2. Rule3, Rule4, Group1, and Group2 become "not selected" due to their associations with Cluster1
3. Group1 becomes "selected", overriding the change to this setting from line 2
4. Value1 changes to using the "sel1" selector
5. Value2 and Value3 change to using the "sel2" selector due to their associations with Cluster2. However, since Value3 does not utilize any selector named "sel2", the effectively associates Value3 with the empty selector. Note that Rule1 is not affected even though it is a member of Cluster2. This is because a refine-value selector only affects Values.
6. The effective value of Value4 changes to "NEWVALUE"
7. Rule2 changes to using the "sel3" selector
8. Rule5 changes to using the "sel1" selector due to its association with Cluster3
9. Rule1 becomes "not selected", overriding the change to this setting from line 1
10. Rule3 becomes "selected", overriding the change to this setting from line 2
11. Value2 and Value3 change to using the "sel5" selector due to their associations with Cluster2. This overrides the change to these settings from line 5.
12. Rule5 changes to using the "sel6" selector, but since it does not utilize any selector named "sel6" this would lead to the use of the empty selector. Since Rule5 does not define any empty selector either, this Rule would effectively not utilize any selectable field. Since the check field is the only selectable field in a Rule, this means that Rule5 would have no check associated with it. (I.e. under evaluation, it would return a result of "notchecked".) This overrides the change to this setting from line 8.

The final configuration of the Benchmark due to application of Profile2 would be as follows:

Item id	Selected?	Applicable Selector
Rule1	not selected	(empty)
Rule2	selected	sel3
Rule3	selected	(empty)
Rule4	not selected	-none-
Rule5	selected	sel6 (Not defined so this becomes -none-)

Group1	selected	-none-
Group2	not selected	-none-
Value1		sel1
Value2		sel5
Value3		sel5
Value4		="NEWVALUE"

Hopefully authors will refrain from creating such a byzantine set of Profiles as described above, but the example does demonstrate how a single Item could be tailored multiple times due to the influence of multiple selectors.

It should also be noted that selectors do not need to be of the same type to override each other's behaviors. All three of the selectors, refine-value, set-value, and set-complex-value, can affect the value or complex-value field of a named Value. However, a Value may have only one value or one complex-value field selected at any time. As a result, the use of any of the aforementioned selectors to change a Value's value/complex-value will replace any prior tailoring of that Value's value or complex-value.

Substitution Processing

XCCDF supports the notion of named parameters, Value objects, which can be set by a user during the tailoring process and then substituted into content specified elsewhere in the Benchmark. XCCDF also supports the notion of plain-text definitions in a Benchmark; these are re-usable chunks of text that may be substituted into other texts using the substitution facilities described here.

As described in the next section, a substitution is always indicated by a reference to the id of a particular Value object, plain-text definition, or other Item in the Benchmark.

During Tailoring and Document Generation, a tool should substitute the title property of the Value object for the reference in any text shown to the user or included in the document. At the tool author's discretion, the title may be followed by the Value object's value property, suitably demarcated. For plain-text definitions, any reference to the definition should be replaced by the string content of the definition.

Any appearance of the instance element in the content of a fix element should be replaced by a locale-appropriate string to represent a target system instance name.

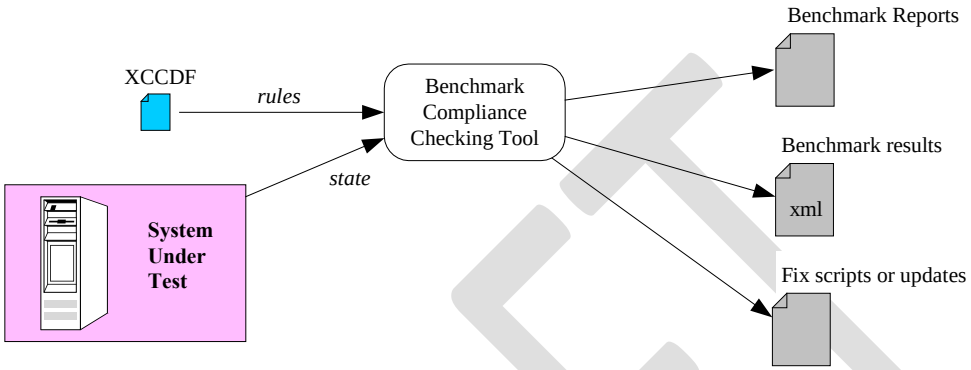
During Compliance Checking, Value objects designated for export to the checking system are passed to it. In general, the interface between the XCCDF checking tool and the underlying checking system or engine must support passing the following properties of the Value: value, type, and operator.

During creation of TestResult objects upon conclusion of Compliance Checking, any fix elements present in applied Rules and matching the platform to which the compliance test was applied should be subjected to substitution and the resulting string used as the value of the fix element for the rule-result element. Each sub element should be replaced by the value of the referenced Value object or plain-text definition actually used during the test. Each instance element should be replaced by the value of the rule-result instance element.

Rule Application and Compliance Scoring

When a Benchmark compliance checking tool performs a compliance run against a system, it accepts as inputs the state of the system and a Benchmark, and produces some outputs, as shown below.

Figure 4 – Workflow for Checking Benchmark Compliance



- **Benchmark Report** – A human-readable report about compliance, including the compliance score, and a listing of which rules passed and which failed on the system. If a given rule applies to multiple parts or components of the system, then multiple pass/fail entries may appear on this list; multiply-instantiated rules are discussed in more detail below. The report may also include recommended steps for improving compliance. The format of the benchmark report is not specified here, but might be some form of formatted or rich text (e.g., HTML).
- **Benchmark results** – A machine-readable file about compliance, meant for storage, long-term tracking, or incorporation into other reports (e.g., a site-wide compliance report). This file may be in XCCDF, using the TestResult object, or it may be in some tool-specific data format.
- **Fix scripts** – Machine-readable files, usually text, the application of which will remediate some or all of the non-compliance issues found by the tool. These scripts may be included in XCCDF TestResult objects.

Rule Check Processing

During the Rule.Content sub-step of the Item Processing Algorithm, the properties of a given Rule are processed. This includes processing of the Rule's check structure(s). This section describes how these structures are processed. Note that, if a Rule contains a complex-check, then the XCCDF interpreter must process this complex-check, ignoring any regular checks that are also contained by the Rule. However, within a given complex-check, processing of component checks follows the same procedures described below.

First, the appropriate set of check structures within the given Rule must be identified. This is done through the selector attribute of the check property. A Profile or manual tailoring action can associate a selector with individual Rules. If a selector name has been associated with a Rule

then all of the Rule's checks whose selector attribute equals that selector name are added to the working set of check structures. If no selector name has been associated with the Rule or, if after this step, the working set remains empty, then all checks with no selector attribute or with a selector attribute that is the empty string are added to the working set of check structures. If the working set of check structures is still empty after this, then no check is processed for the Rule and the Rule will get a result of UNCHECKED. Note that this could happen even when the Rule defines check structures.

Once a working set of check structures has been created, the XCCDF interpreter must select the check to process. At this stage, all check structures in the working set must have different values in their system attributes. An XCCDF interpreter should select the check structure that identifies a check system it supports. If there are multiple checks with supported system attributes, XCCDF interpreters may use their own logic to determine which check to utilize. If the XCCDF interpreter does not support any of the identified checking systems, then no check is processed for this Rule and the Rule will get a result of UNCHECKED.

Now that a single check structure has been selected by the XCCDF interpreter, the interpreter begins processing the check structure itself. First, the check-export statements are processed. For each XCCDF Value named in a check-export statement, determine the appropriate value of this Value. The Value's value might have been set explicitly via a set-value selector in a Profile or from manual tailoring, or it might be identified via a selector name via a refine-value selector in a Profile or also from manual tailoring. The XCCDF interpreter must identify a value for each exported Value.

Finally, a check's check-content-ref and/or check-content structures must be processed. These must be processed in the order in which they appear in the XML, so processing all check-content-ref structures will precede processing of any check-content structure. The XCCDF interpreter must inspect each check-content-ref structure, in order, and determine if content can be retrieved from the given location. If content can be retrieved from the location specified by a given check-content-ref then that content becomes associated with the check and no further check-content-ref or check-content statements are consulted. Only if none of the check-content-ref statements can be resolved to content (or the check contains no check-content-ref statements to begin with) would the body of a check-content statement become the content associated with a given check. If none of a check's check-content-ref statements can be resolved to content and the check has no check-content statement, then no check is associated with this Rule and the Rule will get a result of UNCHECKED. Note that the XCCDF interpreter must make no assertions as to the validity of the content associated with a check. Content that is successfully retrieved from a source might be invalid for a giving checking system, but it is the checking system interpreter's job to catch and report the content error back to the XCCDF interpreter. A check-content-ref would only be skipped if content could not be retrieved from the identified source (such as if a given file could not be found or in the presence of a network failure during a remote reference resolution). Retrieval of invalid content is not grounds for skipping a check-content-ref.

At no point should an XCCDF interpreter "backtrack" in the processing of these steps. For example, once a check with a preferred system is selected by the XCCDF interpreter the interpreter may not attempt to use a different check that uses a different system, even if none of the originally selected check's content can be resolved. Instead, the checking language interpreter's response (probably an error of some kind) must be mapped an appropriate XCCDF result (probably ERROR) and that becomes the result of this check.

Multiply-Instantiated Rules

A security auditor applying a security guidance document to a system typically wants to know two things: how well does the system comply, and how can non-compliant items be reconciled (either fixed or determined not to be salient)?

Many XCCDF documents include Rules that apply to system components. For example, a host OS Benchmark could contain Rules that apply to all users, and a router Benchmark could contain Rules that apply to all network interfaces. When the system holds many of such components, it is not adequate for a tool to inform the administrator or auditor that a Rule failed; it should report exactly which components failed the Rule.

A processing engine that performs a Benchmark compliance test may deliver zero or more pass/fail triples, as described above. In the most common case, each compliance test Rule will yield one result element. In a case where a Rule was applied multiple times to multiple components of the system under test, a single Rule could yield multiple result elements. If the Rule's multiple property is set to true, each instance of the assessment target should be reported and scored separately. Similarly, if a Rule's check field leads to the execution of multiple checks (i.e. a check-content-ref that lacks a name field is used) and the Rule's multi-check property is set to true, each check executed should be reported and scored separately. Otherwise, a Rule contributes to the positive score only if ANDing the results of all instances of that Rule produces a test result of ‘pass’ according to the truth table that appears in the description of the <complex-check> element on page 63. If any component of the target system fails a Rule, then the entire Rule is considered to have failed. This is sometimes called “strict scoring”.

When creating multiple rule-result elements that stem from a single Rule (that is, many target instances and multiple set to true and/or many executed checks and multi-check set to true), all those rule-result elements should identify the same Rule in their rule-idref field. When results are caused by multiple target instances with multiple set to true, the instance field should include, at minimum, the instance name (as described under the description of the Scoring and Results Model below). It may include additional information to provide additional context for that instance. When multiple rule-results are caused by multiple executed checks with multi-check set to true, the rule-result's check field should identify the executed check. This should be done by including a check-content-ref that explicitly names the check executed to produce this particular result. It is possible for a single Rule to reference multiple checks some of which test multiple target instances. This would lead to both the instance and check fields being utilized in the manner described above.

Scoring and Results Model

Semantically, the output or *result* of a single Benchmark compliance test consists of four parts:

1. Rule result list – a vector V of result elements e, with each element a 6-tuple $e=\{r, p, I, t, F, O\}$ where:

- *r* is the Rule id
- *p* is the test result, one of {pass, fail, error, unknown, notapplicable, notchecked, notselected, informational, fixed}. A test whose result *p* is 'error' or 'unknown' is treated as 'fail' for the purposes of scoring; tool developers may wish to alert the user to erroneous and unknown test results. A test whose result *p* is one of {notapplicable, notchecked, informational, notselected} does not contribute to scoring in any way. A test whose result *p* is 'fixed' is treated as a pass for score computation.
- *I* is the instance set, identifying the system components, files, interfaces, or subsystems to which the Rule was applied. Each element of *I* is a triple {n,c,p}, where *n* is the instance name, *c* is the optional instance context, and *p* is the optional parent context. The context *c*, when present, describes the scope or significance of the name *n*. The parent context *p* allows the members of *I* to express nested structure. *I* must be an empty set for tests that are not the result of multiply instantiated Rules (see below).
- *t* is the time at which the result of the Rule application was decided.
- *F* is the set of fixes, from the Rule's fix properties, that should bring the target system into compliance (or at least closer to compliance) with the rule. *F* may be null if the Rule did not possess any applicable fix properties, and must be null when *p* is equal to pass. Each fix *f* in *F* consists of all the properties defined in the description of the Rule fix property: content, strategy, disruption, reboot, system, id, and platform.
- *O* is the set of overrides. Each *o* in *O* consisting of the five properties listed for the rule-result override property: time, authority, old-result, new-result, and remark. Overrides do not affect score computation.

2. Scores – a vector *S*, consisting of one or more score values *s*, with each *s* a pair consisting of a real number and a scoring model identifier.
3. Identification – a vector of strings which identify the Benchmark, Profile (if any), and target system to which the Benchmark was applied.
4. Timestamps – two timestamps recording the beginning and the end of the interval when the Benchmark was applied and the results compiled.

Each element of the pass/fail list *V* conveys the compliance of the system under test, or one component of it, with one Rule of the Benchmark. Each Rule has a weight, title, and other attributes as described above. Each element of *V* may include an instance name, which gives the name of a system component to which the pass or fail designation applies.

XCCDF defines a default scoring model and three optional scoring models, and also permits Benchmark checking tools to support additional proprietary or community models. A Benchmark may specify the scoring model to be used. In the absence of an explicit scoring model specified in the Benchmark, compliance checking tools must compute a score based on the default XCCDF model, and may compute additional scoring values based on other models. The default model computes a score based on relative weights of sibling Rules, as described in the next sub-section.

The fix scripts are collected from the fix properties of the Rules in elements of *V* where *p* is False. A compliance checking or remediation tool may choose to concatenate, consolidate, and/or deconflict the fix scripts; mechanisms for doing so are outside the scope of this specification. In the simplest cases, tools must perform Value substitution on each Rule's fix property before making it part of the output results.

Score Computation Algorithms

This sub-section describes the XCCDF default scoring model, which compliance checking tools must support, and three additional models that tools may support. Each scoring model is identified by a URI. When a Benchmark compliance test is performed, the tool performing the Benchmark may use any score computation model designated by the user. The Benchmark author can suggest or recommend scoring models by indicating them in the Benchmark object using the "model" property. The default model is indicated implicitly for all Benchmarks.

The Default Model

This model is identified by the URI "urn:xccdf:scoring:default". It was the only model supported in XCCDF 1.0, and remains the default for compatibility.

In the default model, computation of the XCCDF score proceeds independently for each collection of siblings in each Group, and then for the siblings within the Benchmark. This relative-to-siblings weighted scoring model is designed for flexibility and to foster independent authorship of collections of Rules. Benchmark authors must keep the model in mind when assigning weights to Groups and Rules.

The objects of an XCCDF Benchmark form the nodes of a tree. The default model score computation algorithm simply computes a normalized weighted sum at each tree node, omitting Rules and Groups that are not selected and Groups that have no selected Rules under them. The algorithm at each selected node is:

Sub-Step	Description
Score.Rule	If the node is a Rule, then assign a count of 1, and if the test result is 'pass', assign the node a score of 100, otherwise assign a score of 0.
Score.Group.Init	If the node is a Group or the Benchmark, assign a count of 0, a score *s* of 0.0, and an accumulator *a* of 0.0.
Score.Group.Recurse	For each selected child of this Group or Benchmark, do the following: (1) compute the count and weighted score for the child using this algorithm, (2) if the child's count value is not 0, then add the child's weighted score to this node's score *s*, add 1 to this node's count, and add the child's weight value to the accumulator *a*.
Score.Group.Normalize	Normalize this node's score: compute $s = s / a$.
Score.Weight	Assign the node a weighted score equal to the product of its score and its weight.

The final test score is the normalized score value on the root node of the tree, which is the Benchmark object.

The Flat Model

This model is identified by the URI "urn:xccdf:scoring:flat".

Under this model, the set of Rule results is treated as a vector *V*, as described above. The following algorithm is used to compute the score.

Sub-Step	Description
Score.Init	Initialize both the score *s* and the maximum score *m* to 0.0.
Score.Rules	For each element *e* in *V* where *e.p* is not a member of the set {notapplicable, notchecked, informational, notselected}: - add the weight of rule *e.r* to *m* - if the value *e.p* equals 'pass' or 'fixed', add the weight of the rule *e.r* to *s*.

Thus, the flat model simply computes the sum of the weights for the Rules that passed as the score, and the sum of the weights of all the applicable Rules as the maximum possible score. This model is simple and easy to compute, but scores between different target systems may not be directly comparable because the maximum score can vary.

The Flat Unweighted Model

This model is identified by the URI "urn:xccdf:scoring:flat-unweighted". It is computed in exactly the same way as the flat model, except that all weights not set to 0 are taken to be 1.0. Items with weights of 0 remain 0 in this model and, as such, do not contribute to the final score.

The Absolute Model

This model is identified by the URI "urn:xccdf:scoring:absolute". It gives a score of 1 only when all applicable Rules in the benchmark pass. It is computed by applying the Flat Model and returning 1 if *s*=*m*, and 0 otherwise.

4. XML Representation

This section defines a concrete representation of the XCCDF data model in XML, using both core XML syntax and XML Namespaces.

4.1. XML Document General Considerations

The basic document format consists of a root "Benchmark" element, representing a Benchmark object. Its child elements are the contents of the Benchmark object, as described in Section 3.2.

All the XCCDF elements in the document will belong to the XCCDF namespace, including the root element. The namespace URI corresponding to this version of the specification is "http://checklists.nist.gov/xccdf/1.2". The namespace of the root Benchmark element serves to identify the XCCDF version for a document. Applications that process XCCDF can use the namespace URI to decide whether or not they can process a given document. If a namespace prefix is used, the suggested prefix string is "cdf".

XCCDF attributes are not namespace qualified. All attributes begin with a lowercase letter, except the "Id" attribute (for compatibility with XML Digital Signatures [10]).

The example below illustrates the outermost structure of an XCCDF XML document.

Example 5 – Top-Level XCCDF XML

```
<?xml version="1.0" ?>
<cdf:Benchmark  id="example1"  xml:lang="en"  Id="toSign"
      xmlns:htm="http://www.w3.org/1999/xhtml"
      xmlns:cdf="http://checklists.nist.gov/xccdf/1.2"
      xmlns:cpe="http://cpe.mitre.org/dictionary/2.0"/>
  <cdf:status date="2010-06-01">draft</cdf:status>
  <cdf:title>Example Benchmark File</cdf:title>
  <cdf:description>
     A <htm:b>Small</htm:b> Example
  </cdf:description>
  <cdf:platform idref="cpe:/o:cisco:ios:12.0"/>
  <cdf:version>0.2</cdf:version>
  <cdf:reference href="http://www.ietf.org/rfc/rfc822.txt">
      Standard for the Format of ARPA Internet Text Messages
  </cdf:reference>
</cdf:Benchmark>
```

Validation is strongly suggested but not required for tools that process XCCDF documents. The XML Schema attribute 'schemaLocation' may be used to refer to the XCCDF Schema (see Appendix A).

Properties of XCCDF objects marked as type 'HTML-enabled text' in Section 3.2 may contain embedded formatting, presentation, and hyperlink structure. XHTML Basic tags must be used to express the formatting, presentation, and hyperlink structure within XCCDF documents. In particular, the core modules noted in the XHMTL Basic Recommendation [4] are permitted in XCCDF documents, plus the Image module and the Presentation module. How an XCCDF processing tool handles embedded XHTML content in XCCDF text properties is

implementation-dependent, but at the least every tool must be able to process XCCDF files even when embedded XHTML elements are present. Tools that perform document generation processing should attempt to preserve the formatting semantics implied by the Text and List modules, support the link semantics implied by the Hypertext module, and incorporate the images referenced via the Image module. Such tools may also wish to establish conventions for each of the or class attribute values, as described in section 3.2.

4.2. XML Element Dictionary

This subsection describes each of the elements and attributes of the XCCDF XML specification. Each description includes the parent elements feasible for that element, as well as the child elements it might normally contain. Most elements are in the XCCDF namespace, which for version 1.2 is "http://checklists.nist.gov/xccdf/1.2". The full schema appears in Appendix A.

Many of the elements listed below are described as containing formatted text (type 'HTML-enabled text' in Section 3.2). These elements may contain Value substitutions, and formatting expressed as described in Section 4.3.

XML is case-sensitive. The XML syntax for XCCDF follows a common convention for representing object-oriented data models in XML: elements that correspond directly to object classes in the data model have names with initial caps. Mandatory attributes and elements are shown in bold. Child elements are listed in the order in which they must appear. Elements which are not part of the XCCDF namespace are shown in italics.

<Benchmark>

This is the root element of the XCCDF document; it must appear exactly once. It encloses the entire Benchmark, and contains both descriptive information and Benchmark structural information. The id attribute must be a unique identifier.

Content:	elements
Cardinality:	1
Parent Element:	*none*
Attributes:	**id**, resolved, style, style-href, xml:lang, Id (note: "Id" is needed only for digital signature security)
Child Elements:	**status**,dc-status **title**, description, notice, front-matter, rear-matter, reference, *platform-specification*, platform, version, metadata, Profile, Value, Group, Rule, signature

Note that the order of Group and Rule child elements may matter for the appearance of a generated document. Group and Rule children may be freely intermingled, but they must appear after any Value children. All the other children must appear in the order shown, and multiple instances of a child element must be adjacent.

<Group>

A Group element contains descriptive information about a portion of a Benchmark, as well as Rules, Values, and other Groups. A Group must have a unique id attribute to be referenced from

other XCCDF documents. The 'hidden' and 'allowChanges' attributes are of boolean type and default to false. The weight attribute is a positive real number.

Content:	elements
Cardinality:	0-n
Parent Elements:	Benchmark, Group
Attributes:	**id**, cluster-id, extends ***(deprecated)***, abstract ***(deprecated)***, hidden, prohibitChanges, selected, weight, Id
Child Elements:	status, dc-status, version, title, description, warning, question, reference, rationale, platform, requires, conflicts, Value, Group, Rule, metadata, signature

All child elements are optional, but every group should have a title as this will help human editors and readers understand the purpose of the Group. Group and Rule children may be freely intermingled. All the other children must appear in the order shown, and multiple instances of a child element must be adjacent.

Although Groups have extends and abstract attributes, use of these attributes is now deprecated.

<Profile>

A Profile element encapsulates a tailoring of the Benchmark. It consists of an id, descriptive text properties, and zero or more selectors that refer to Group, Rule, and Value objects in the Benchmark. There are five selector elements: select, set-value, set-complex-value, refine-rule, and refine-value.

Profile elements may only appear as direct children of the Benchmark element. A Profile may be defined as extending another Profile, using the 'extends' attribute.

Content:	elements
Cardinality:	0-n
Parent Elements:	Benchmark
Attributes:	abstract, **id**, extends, prohibitChanges, Id, note-tag
Child Elements:	status, dc-status, version, **title**, description, reference, platform,select, set-value, set-complex-value, refine-value, refine-rule, metadata, signature

Profiles are designed to support encapsulation of a set of tailorings. A Profile implicitly includes all the Groups and Rules in the Benchmark, and the select element children of the Profile affect which Groups and Rules are selected for processing when the Profile is in effect. The example below shows a very simple Profile.

Example 8 – Example of a Simple XCCDF Profile

```
<cdf:Profile id="strict" prohibitChanges="1"
```

```
            extends="lenient" note-tag="strict-tag">
  <cdf:title>Strict Security Settings</cdf:title>
  <cdf:description>
    Strict lockdown rules and values, for hosts deployed to
    high-risk environments.
  </cdf:description>
  <cdf:set-value idref="password-len">10</cdf:set-value>
  <cdf:refine-value idref="session-timeout" selector="quick"/>
  <cdf:refine-rule idref="session-auth-rule" selector="harsh"/>
  <cdf:select idref="password-len-rule" selected="1"/>
  <cdf:select idref="audit-cluster" selected="1"/>
  <cdf:select idref="telnet-disabled-rule" selected="1"/>
  <cdf:select idref="telnet-settings-cluster" selected="0"/>
</cdf:Value>
```

<Rule>

A Rule element defines a single Item to be checked as part of a Benchmark, or an extendable base definition for such Items. A Rule must have a unique id attribute, and this id is used when the Rule is used for extension, referenced from Profiles, or referenced from other XCCDF documents.

The 'extends' attribute, if present, must have a value equal to the id attribute of another Rule. The 'weight' attribute must be a positive real number. Rules may not be nested.

Content:	elements
Cardinality:	0-n
Parent Elements:	Benchmark, Group
Attributes:	**id**, cluster-id, extends, hidden, multiple, multi-check, prohibitChanges, role ***(deprecated)***, selected, severity**,** weight, Id
Child Elements:	status, dc-status, version, title, description, warning, question, reference, rationale, platform, requires, conflicts, ident, impact-metric ***(deprecated)***, profile-note, fixtext, fix, complex-check, check, metadata, signature

The check child element of a Rule is the vital piece that specifies how to check compliance with a security practice or guideline. See the description of the check element below for more information. Example 6 shows a very simple Rule element.

Example 6 – A Simple XCCDF Rule

```
<cdf:Rule id="pwd-perm" selected="1" weight="6.5" severity="high">
  <cdf:title>Password File Permission</cdf:title>
  <cdf:description>Check the access control on the password
     file.  Normal users should not be able to write to it.
  </cdf:description>
  <cdf:requires idref="passwd-exists"/>
  <cdf:fixtext>
```

```
    Set permissions on the passwd file to owner-write, world-read
  </cdf:fixtext>
  <cdf:fix strategy="restrict" reboot="0" disruption="low">
       chmod 644 /etc/passwd
  </cdf:fix>
  <cdf:check system="http://www.mitre.org/XMLSchema/oval">
    <cdf:check-content-ref href="ovaldefs.xml" name="OVAL123"/>
  </cdf:check>
</cdf:Rule>
```

An XCCDF Rule may only extend a Rule that is within its visible scope. The visible scope includes sibling Rules and Rules that are siblings of ancestor Groups.

Circular dependencies of extension may not be defined.

<TestResult>

The TestResult object encapsulates the result of applying a Benchmark to one target system. The TestResult element normally appears as the child of the Benchmark element, although it may also appear as the top-level element of a file.

Content:	elements
Cardinality:	0-n
Parent Elements:	Benchmark
Attributes:	**id**, start-time, **end-time**, Id
Child Elements:	title, remark, organization, identity, profile, set-value, set-complex-value, target, target-address, target-facts, rule-result, score, metadata, signature

The id attribute is a mandatory unique-id for a test result. The start-time and end-time attributes must have the format of a timestamp; the end-time attribute is mandatory, and gives the time that the application of the Benchmark completed.

The example below shows a TestResult object with a few rule-result children.

Example 9 – Example of XCCDF Benchmark Test Results

```
<cdf:TestResult id="ios-test5" end-time="2007-09-25T7:45:02-04:00"
  xmlns:cdf="http://checklists.nist.gov/xccdf/1.2">
   <cdf:benchmark href="ios-sample-12.4.xccdf.xml"/>
   <cdf:title>Sample Results Block</cdf:title>
   <cdf:remark>Test run by Bob on Sept 25, 2007</cdf:remark>
   <cdf:organization>Department of Commerce</cdf:organization>
   <cdf:organization>National Institute of Standards and Technology
   </cdf:organization>
   <cdf:identity authenticated="1" privileged="1">admin_bob
```

```
   </cdf:identity>
   <cdf:target>lower.test.net</cdf:target>
   <cdf:target-address>192.168.248.1</cdf:target-address>
   <cdf:target-address>2001:8::1</cdf:target-address>
   <cdf:target-facts>
        <cdf:fact type="string"
         name="urn:xccdf:fact:ethernet:MAC">
            02:50:e6:c0:14:39
        </cdf:fact>
        <cdf:fact name="urn:xccdf:fact:OS:IOS">1</cdf:fact>
   </cdf:target-facts>
   <cdf:set-value idref="exec-timeout-time">10</cdf:set-value>
   <cdf:rule-result idref ="ios12-no-finger-service"
                    time="2007-09-25T13:45:00-04:00">
        <cdf:result>pass</cdf:result>
   </cdf:rule-result>
   <cdf:rule-result idref ="req-exec-timeout"
                    time="2007-09-25T13:45:06-04:00">
        <cdf:result>fail</cdf:result>
        <cdf:instance>console</cdf:instance>
        <cdf:fix>
             line console
             exec-timeout 10 0
        </cdf:fix>
   </cdf:rule-result>
   <cdf:score>67.5</cdf:score>
   <cdf:score system="urn:xccdf:scoring:absolute">0</cdf:score>
</cdf:TestResult>
```

<Value>

A Value element represents a named parameter whose title or value may be substituted into other strings in the Benchmark (depending on the form of processing to which the Benchmark is being subjected), or it may represent a basis for the definition of such parameters via extension. A Value object must have a unique id attribute to be referenced for substitution or extension or for inclusion in another Benchmark.

A Value object may appear as a child of the Benchmark, or as a child of a Group. Value objects may not be nested. The value and default child elements must appear first.

Content:	elements
Cardinality:	0-n
Parent Elements:	Benchmark, Group
Attributes:	**id**, cluster-id, extends, hidden, prohibitChanges, operator, type, interactive, interfaceHint, Id
Child Elements:	status, dc-status, version, title, description, warning, question, reference, value, complex-value, default, complex-default, match, lower-bound, upper-bound, choices, source, metadata, signature

In the absence of any Profile or tailoring actions, the default value property in a Value is the one with an empty or absent selector. If there is no value property with an empty or absent selector, the first value property in top-down processing of the XML will be the default property. For all other selectable Value properties (i.e. default, match, upper-bound, lower-bound, and choices), the default activity will be to ignore all properties without an empty or absent selector.

The type attribute is optional, but if it appears it must be one of 'number', 'string', or 'boolean'. A tool performing tailoring processing may use this type name to perform user input validation. Example 7, below, shows a very simple Value object.

Example 7 – Example of a Simple XCCDF Value

```
<cdf:Value id="web-server-port" type="number" operator="equals">
  <cdf:title>Web Server Port</cdf:title>
  <cdf:description>TCP port on which the server listens
  </cdf:description>
  <cdf:value>12080</cdf:value>
  <cdf:default>80</cdf:default>
  <cdf:lower-bound>0</cdf:lower-bound>
  <cdf:upper-bound>65535</cdf:upper-bound>
</cdf:Value>
```

(Note that the match element applies only for validation during XCCDF tailoring, while the operator attribute applies only for rule checking. People often confuse these.)

A Value object must contain at least one of either a value or complex-value child element. It does not require both, although that is also permissible. The complex-value and complex-default child elements are used to encapsulate complex structures, such as lists or externally defined data types. It is permissible for a single Value to allow both simple (e.g. value) and complex (e.g. complex-value) structures as options for tailoring.

<benchmark>

This simple element may only appear as the child of a TestResult. It indicates the Benchmark for which the TestResult records results. It has two attributes: href, which gives the URI of the Benchmark XCCDF document, and id, which holds the value of that Benchmark's id property. It must be an empty element.

Content	*none*
Cardinality:	0-1
Parent Elements:	TestResult
Attributes:	**href**, id
Child Elements:	*none*

The Benchmark element must be used in a standalone TestResult file (an XCCDF document file whose root element is TestResult).

<check>

This element holds a specification for how to check compliance with a Rule. It may appear as a child of a Rule element, or in somewhat abbreviated form as a child of a rule-result element inside a TestResult object.

The child elements of the check element specify the values to pass to a checking engine and the logic for the checking engine to apply. The logic may be embedded directly as inline text or XML data, or may be a reference to an element of an external file indicated by a URI. If the compliance checking system uses XML namespaces, then the system attribute for the system should be its namespace. The default or nominal content for a check element is a compliance test expressed as an OVAL Definition or a reference to an OVAL Definition, with the system attribute set to the OVAL namespace.

The check element may also be used as part of a TestResult rule-result element; in that case it holds or refers to the check run and/or detailed output from the checking engine.

Content:	elements
Cardinality:	0-n
Parent Elements:	Rule, rule-result
Attributes:	id, negate, selector, **system**
Child Elements:	check-import, check-export, check-content-ref, check-content

A check element may have a selector attribute, which may be referenced from a Benchmark Profile as a means of refining the application of the Rule. When Profiles are not used, then all check elements with non-empty selectors are ignored. Check elements may also have a negate attribute. If set to true, then the final result of this check is negated according to the truth tables that appear on page 63.

Several check elements may appear as children of the same Rule element. Sibling check elements must have different values for the combination of their selector and system attributes, and different values for their id attribute (if any). A tool processing the Benchmark for compliance checking must pick at most one check or complex-check element to process for each Rule.

The check element may contain zero or more check-import elements, followed by zero or more check-export elements, followed by zero or more check-content-ref elements, followed by at most one check-content element. If two or more check-content-ref elements appear, then they represent alternative locations from which a tool may obtain the check content. Tools should process the alternatives in the order in which they appear in the XML. The first check-content-ref from which content can be successfully retrieved should be used. (Note that ensuring the validity of this content is not the responsibility of an XCCDF interpreter.) If both check-content-ref elements and check-content elements appear, tools should use the check-content only if all references cannot be resolved to provide content.When a check element is a child of a Rule object, the check-import element must be empty. When a check element is a child rule-result object, check-import elements contain the value retrieved from the checking system.

<check-import>

This element identifies a value to be retrieved from the checking system during testing of a target system. The import-xpath holds an optional XPath 1.0 string that can further narrow the selection of data associated with the item identified in the import-name attribute.

Content:	mixed
Cardinality:	0-n
Parent Elements:	check
Attributes:	**import-name**, import-xpath
Child Elements:	*any*

When a check-import element appears in the context of a Rule object, it must be empty. When it appears in the context of a rule-result, its content is the value retrieved from the checking system.

When a check-import element is given in a Rule, after the associated check results have been collected, the result structure returned by the checking engine is processed to collect the named information. This information is then recorded in the check-import element in the corresponding rule-result. This could be a single value (for example, if an OVAL Variable were named the result check-import would contain the Variable value at the time the named OVAL Definition was evaluated) or structured XML (for example, if an OVAL Object were named the OVAL System Characteristic Items would be listed). In the latter case, the import-xpath can be used to traverse the structured XML and identify specific information of interest to record in the result.

<check-export>

This specifies a mapping from an XCCDF Value object to a checking system variable. The value-id attribute must match the id attribute of a Value object in the Benchmark.

Content:	none
Cardinality:	0-n
Parent Elements:	check
Attributes:	**value-id**, **export-name**
Child Elements:	*none*

<check-content>

This element holds the actual code of a Benchmark compliance check, in the language or system specified by the check element's system attribute. At least one of check-content or check-content-ref must appear in each check element. The body of this element can be any XML, but cannot contain any XCCDF elements. XCCDF tools are not required to process this element; typically it will be passed to a checking system or engine.

Content:	*any non-XCCDF*
Cardinality:	0-1

Parent Elements:	check
Attributes:	*none*
Child Elements:	*special*

<check-content-ref>

This element points to a Benchmark compliance check, in the language or system specified by the check element's system attribute. At least one of check-content or check-content-ref must appear in each check element. The 'href' attribute identifies the document, and the optional name attribute may be used to refer to a particular part, element, or component of the document. The default behavior of a check-content-ref that does not have a name attribute is to execute all checks in the referenced file and AND their results together into a single XCCDF rule-result. However, if a Rule's multi-check attribute is set to true and a nameless check-content-ref is used, each check in the targeted file should be reported as a separate XCCDF rule-result.

Content:	*none*
Cardinality:	0-n
Parent Elements:	check
Attributes:	**href**, name
Child Elements:	*none*

<choices>

The choices element may be a child of a Value, and it enumerates one or more legal values for the Value. If the boolean 'mustMatch' attribute is true, then the list represents all the legal values; if mustMatch is absent or false, then the list represents suggested values, but other values might also be legal (subject to the parent Value's upper-bound, lower-bound, or match attributes). The choices element may have a selector attribute that is used for tailoring via a Profile. The list given by this element is intended for use during tailoring and document generation; it has no role in Benchmark compliance checking. At least one instance of choice or complex-choice must appear as a child of this element.

Content:	elements
Cardinality:	0-n
Parent Elements:	Value
Attributes:	mustMatch, selector
Child Elements:	choice, complex-choice

<choice>

This string element is used to hold a possible legal value for a Value object. It or a complex-choice element must appear as the child of a choices element, and has no attributes or child elements.

Content:	string
Cardinality:	1-n
Parent Elements:	choices
Attributes:	*none*
Child Elements:	*none*

If a tool presents the choice values from a choices element to a user, they should be presented in the order in which they appear.

<complex-check>

This element may only appear as a child of a Rule. It contains a boolean expression composed of operators (and, or, not) and individual checks.

Content:	elements
Cardinality:	0-1
Parent Elements:	Rule
Attributes:	**operator**, negate
Child Elements:	complex-check, check

Truth tables for boolean operation in complex checks are given below; all the abbreviations in the truth tables come from the description of the TestResult/rule-result object (see page 34).

With an "AND" operator, the complex-check evaluates to Pass only if all of its enclosed terms (checks and complex-checks) evaluate to Pass. For purposes of evaluation, Pass (P) and Fixed (X) are considered equivalent. The truth table for "AND" is given below.

AND	P	F	U	E	N	K	S	I
P	P	F	U	E	P	P	P	P
F	F	F	F	F	F	F	F	F
U	U	F	U	U	U	U	U	U
E	E	F	U	E	E	E	E	E
N	P	F	U	E	N	N	N	N
K	P	F	U	E	N	K	K	K
S	P	F	U	E	N	K	S	S
I	P	F	U	E	N	K	S	I

The 'OR' operator evaluates to Pass if any of its enclosed terms evaluate to Pass. The truth table for "OR" is given below.

OR	P	F	U	E	N	K	S	I
P	P	P	P	P	P	P	P	P
F	P	F	U	E	F	F	F	F
U	P	U	U	U	U	U	U	U

E	P	E	U	E	E	E	E	E
N	P	F	U	E	N	N	N	N
K	P	F	U	E	N	K	K	K
S	P	F	U	E	N	K	S	S
I	P	F	U	E	N	K	S	I

If the negate attribute is set to true, then the result of the complex-check must be complemented (inverted). The full truth table for negation is given below.

	P	F	U	E	N	K	S	I
not	F	P	U	E	N	K	S	I

The example below shows a complex-check with several components.

Example 10 – Example of XCCDF Complex Check

```
<cdf:complex-check operator="OR">
  <cdf:check system="http://oval.mitre.org/XMLSchema/oval">
     <cdf:check-content-ref href="xpDefs.xml" name="XP-P1"/>
  </cdf:check>
  <cdf:complex-check operator="AND" negate="1">
     <cdf:check system="http://oval.mitre.org/XMLSchema/oval">
         <cdf:check-content-ref href="xpDefs.xml" name="XP-CX"/>
     </cdf:check>
     <cdf:check system="http://www.cisecurity.org/xccdf/interactive/1.0">
         <cdf:check-content-ref href="xpInter.xml" name="XP-6"/>
     </cdf:check>
   </cdf:complex-check>
</cdf:complex-check>
```

<complex-choice>

This element is used to hold a possible legal complex value for a Value object. It or a choice element must appear as the child of a choices element, and has no attributes. At least one of item (for a list) or external-type (for one or more instances of an externally defined data type) must appear as a child of this element.

Content:	elements
Cardinality:	0-n (at least one of choice or complex-choice must be present in each choices element)
Parent Elements:	choices
Attributes:	*none*
Child Elements:	item, external-type

<complex-default>

This element is used to hold the default or reset value of a Value object. It may only appear as a child of a Value element. This element may have a selector attribute, which may be used to

designate different defaults for different Benchmark Profiles. If more than one default and/or complex-default element appears as the child of a Value, at most one of these properties may omit the selector attribute. At least one of item (for a list) or external-type (for one or more instances of an externally defined data type) must appear as a child of this element.

Content:	elements
Cardinality:	0-n
Parent Elements:	Value
Attributes:	selector
Child Elements:	item, external-type

<complex-value>

This element is used to hold complex structures to serve as the value of a Value object. It or a value element must appear as the child of a Value element. This element may have a selector tag attribute, which identifies it for Value refinement by a Profile. This element may appear more than once, but at most one of the sibling instances of this element or sibling value elements may omit the selector tag. At least one of item (for a list) or external-type (for one or more instances of an externally defined data type) must appear as a child of this element.

Content:	elements
Cardinality:	0-n (at least one of value or complex-value must be present in each Value)
Parent Elements:	Value
Attributes:	selector
Child Elements:	item, external-type

<conflicts>

The conflicts element may be a child of any Group or Rule and it specifies the id property of another Group or Rule item whose selection conflicts with this one. Each conflicts element specifies a single conflicting Item using its 'idref' attribute; if the semantics of the Benchmark need multiple conflicts, then multiple conflicts elements may appear. A conflicts element must be empty.

Content:	*none*
Cardinality:	0-n
Parent Elements:	Group, Rule
Attributes:	**idref**
Child Elements:	*none*

<cpe-list>

This element holds names and descriptions for one or more platforms, using the XML schema defined for the Common Platform Enumeration (CPE) 1.0. CPE Names are URIs, and may be

used for all platform identification in an XCCDF document. This element is deprecated, and appears in the XCCDF specification only for compatibility with earlier versions.

Content:	elements *(from the CPE 1.0 dictionary namespace)*
Cardinality:	0-1
Parent Elements:	Benchmark
Attributes:	*none*
Child Elements:	*cpe-item*

<dc-status>

The dc-status field allows additional status information to be associated with Benchmarks, Groups, Rules, Values, or Profiles. This status information should be expressed as elements of the Dublin Core Metadata Initiative (DCMI) Simple DC Element specification, as described in [12] and [14]. This element is not inherited.

Content:	element
Cardinality:	0-1
Parent Elements:	Benchmark, Group, Rule, Value, Profile
Attributes:	*none*
Child Elements:	*Dublin Core elements*

<default>

This string element is used to hold the default or reset value of a Value object. It may only appear as a child of a Value element, and has no child elements. This element may have a selector attribute, which may be used to designate different defaults for different Benchmark Profiles. If more than one default and/or complex-default element appears as the child of a Value, at most one of these properties may omit the selector attribute. Note that this property holds the default value displayed to the user by tools during tailoring, not the default value of a Value.

Content:	string
Cardinality:	0-n
Parent Elements:	Value
Attributes:	selector
Child Elements:	*none*

<description>

This element provides the descriptive text for a Benchmark, Rule, Group, or Value. It has two attributes: xml:lang and override. Multiple description elements may appear with different values for their xml:lang attribute (see also next section).

Content:	mixed
Cardinality:	0-n
Parent Elements:	Benchmark, Group, Rule, Value, Profile
Attributes:	xml:lang, override
Child Elements:	sub, *xhtml elements*

The 'sub' element may appear inside a description, and in many other descriptive text elements. During document generation, each instance of the 'sub' element should be replaced by the title of the Item or other object to which it refers. For more information, see page 46.

<external-type>

This element holds arbitrary XML representing an instance of an externally defined data type. The body of this element can be any XML, but cannot contain any XCCDF elements. The author must identify the namespace of the external data type structures in order to support validation. Any number of externally defined XML structures may appear within an instance of external-type.

Content:	elements
Cardinality:	0-1 (at least one of external-type or item must be present in the parent element)
Parent Elements:	complex-value, complex-default, set-complex-value, complex-choice
Attributes:	*none*
Child Elements:	*special*

<fact>

This element holds a single type-name-value fact about the target of a test. The name is a URI. Pre-defined names start with "urn:xccdf:fact", but tool developers may define additional platform-specific and tool-specific facts.

Content:	string
Cardinality:	0-n
Parent Elements:	target-facts
Attributes:	**name**, type
Child Elements:	*none*

The following types are supported: "number", "string", and "boolean" (the default).

<fix>

This element may appear as the child of a Rule element or a rule-result element. When it appears as a child of a Rule element, it contains string data for a command, script, or procedure

that should bring the target into compliance with the Rule. It may not contain XHTML formatting. The fix element may contain XCCDF Value substitutions specified with the sub element or instance name substitution specified with an instance element.

Content	mixed
Cardinality:	0-n
Parent Elements:	Rule, rule-result
Attributes:	id, complexity, disruption, platform, reboot, strategy, system
Child Elements:	instance, sub

The fix element supports several attributes that the Rule author can use to provide additional information about the remediation that the fix element contains. The attributes and their permissible values are listed below.

Attribute	Values
`id`	A local id for the fix, which allows fixtext elements to refer to it. These need not be unique; several fix elements might have the same id but different values for the other attributes.
`complexity`	A keyword that indicates the complexity or difficulty of applying the fix to the target. Allowed values: • `unknown` – default, complexity not defined • `low` – the fix is very simple to apply • `medium` –fix is moderately difficult or complex • `high` – the fix is very complex to apply
`disruption`	A keyword that designates the potential for disruption or degradation of target operation. Allowed values: • `unknown` – default, disruption not defined • `low` – little or no disruption expected • `medium` – potential for minor or short-lived disruption • `high` – potential for serious disruption
`platform`	A platform identifier; this should appear on a fix when the content applies to only one platform out of several to which the Rule could apply.
`reboot`	Boolean – if remediation will require a reboot or hard reset of the target (‘1’ means reboot required)

Attribute	Values
strategy	A keyword that designates the approach or method that the fix uses. Allowed values: • **unknown** – default, strategy not defined • **configure** – adjust target configuration/settings • **patch** – apply a patch, hotfix, update, etc. • **disable** – turn off or uninstall a target component • **enable** – turn on or install a target component • **restrict** – adjust permissions, access rights, filters, or other access restrictions • **policy** – remediation requires out-of-band adjustments to policies or procedures • **combination** – strategy is a combination or two or more approaches
system	A URI that identifies the scheme, language, or engine for which the fix is written. Several general URIs are defined, but platform-specific URIs may be expected. (For a list of pre-defined fix system URIs, see Appendix C.)

The platform attribute defines the platform for which the fix is intended, if its parent Rule applied to multiple platforms. The value of the platform attribute should be one of the platform strings defined for the Benchmark. If the fix's platform attribute is not given, then the fix applies to all platforms to which its enclosing Rule applies.

As a special case, fix elements may also appear as children of a rule-result element in a TestResult. In this case, the fix element should not any child elements; its content should be a simple string. When a fix element is the child of rule-result, it is assumed to have been 'instantiated' by the testing tool, and any substitutions or platform selection already made.

<fixtext>

This element, which may only appear as a child of a Rule element, provides text that explains how to bring a target system into compliance with the Rule. Multiple instances may appear in a Rule, with different attribute values.

Content:	mixed
Cardinality:	0-n
Parent Elements:	Rule
Attributes:	xml:lang, fixref, disruption, reboot, strategy, override
Child Elements:	sub, *xhtml elements*

The fixtext element and its counterpart, the fix element, are fairly complex. They can accept a number of attributes that describe aspects of the remediation. The xml:lang attribute designates the locale for which the text was written; it is expected that fix elements usually will be locale-

independent. The following attributes may appear on the fixtext element (for details about most of them, refer to the table under the fix element definition, p. 66).

Attribute	Values
`fixref`	A reference to the id of a fix element
`complexity`	A keyword that indicates the difficulty or complexity of applying the described fix to the target
`disruption`	A keyword that designates the potential for disruption or degradation of target operation
`reboot`	Boolean – if the remediation described in the fixtext will require a reboot or reset of the target
`strategy`	A keyword that designates the approach or method that the fix uses

The fixtext element may contain XHTML elements to aid in formatting.

<front-matter>

This element contains textual content intended for use during Document Generation processing only; it is introductory matter that should appear at or near the beginning of the generated document. Multiple instances may appear with different xml:lang values.

Content:	mixed
Cardinality:	0-n
Parent Elements:	Benchmark
Attributes:	xml:lang
Child Elements:	sub, *xhtml elements*

<ident>

This element contains a string (name) which is a long-term globally meaningful identifier in some naming scheme. The content of the element is the name, and the system attribute contains a URI which designates the organization or scheme that assigned the name (see Appendix C for assigned URIs).

Content:	string
Cardinality:	0-n
Parent Elements:	Rule, rule-result
Attributes:	**system**
Child Elements:	*none*

<identity>

This element may appear only as a child of a TestResult. It provides up to three pieces of information about the system identity or user employed during application of the Benchmark: whether the identity was authenticated, whether the identity was granted administrative or other special privileges, and the name of the identity.

Content:	string
Cardinality:	0-1
Parent Elements:	TestResult
Attributes:	**authenticated, privileged**
Child Elements:	*none*

The attributes are both required, and both boolean. The string content of the element is the identity name, and may be omitted.

<impact-metric>

This element contains a string representation of the potential impact of failure to conform to a Rule. The content must be a CVSS base vector, expressed using the format defined in the CVSS 2.0 specification [18]. This element is deprecated, and appears in this specification only for backward compatibility.

Content:	string
Cardinality:	0-1
Parent Elements:	Rule
Attributes:	*none*
Child Elements:	*none*

<instance>

The instance element may appear in two situations. First, it may appear as part of a TestResult, as a child of a rule-result element; in that situation it contains the name of a target component to which a Rule was applied, in the case of multiply-instantiated rules.

Content:	string
Cardinality:	0-n
Parent Elements:	rule-result
Attributes:	context, parentContext
Child Elements:	*none*

If the context attribute is omitted, the value of the context defaults to “undefined”. At most one instance child of a rule-result may have a context of “undefined”.

Second, the instance element may appear as part of a Rule, as a child of the fix element. In that situation it represents a place in the fix text where the name of a target component should be substituted, in the case of multiply-instantiated rules.

Content:	*none*
Cardinality:	0-n
Parent Elements:	fix
Attributes:	context
Child Elements:	*none*

If the context attribute is omitted, the value of the context defaults to "undefined".

<item>

This string element is used to hold a single simple (number, string, or boolean) value. It represents one item in a list. It has no child elements or attributes.

Content:	string
Cardinality:	0-n (at least one of external-type or item must be present in the parent element)
Parent Elements:	complex-value, complex-default, set-complex-value, complex-choice
Attributes:	*none*
Child Elements:	*none*

<lower-bound>

This element may appear zero or more times as a child of a Value element. It is used to constrain value input during tailoring, when the Value's type is "number". It contains a number; values supplied by the user for tailoring the Benchmark must be no less than this number. This element may have a selector tag attribute, which identifies it for Value refinement by a Profile. If more than one lower-bound element appears as the child of a Value, at most one of them may omit the selector attribute.

Content:	number
Cardinality:	0-n
Parent Elements:	Value
Attributes:	selector
Child Elements:	*none*

<match>

This element may appear zero or more times as a child of a Value element. It is used to constrain value input during tailoring. It contains a regular expression that a user's input for the

value must match. This element may have a selector tag attribute, which identifies it for Value refinement by a Profile. If more than one match element appears as the child of a Value, at most one of them may omit the selector attribute.

Content:	string
Cardinality:	0-n
Parent Elements:	Value
Attributes:	selector
Child Elements:	*none*

<message>

This element is optional, but may appear one or more times as a child of a rule-result element inside a TestResult object. It holds a single informational or error message that was returned by the checking engine.

Content:	string
Cardinality:	0-n
Parent Elements:	rule-result
Attributes:	**severity**
Child Elements:	*none*

The severity attribute denotes the seriousness or conditions of the message. There are three message severity values: “error”, “warning”, and “info”. These elements do not affect scoring; they are present merely to convey diagnostic information from the checking engine. XCCDF tools that deal with TestResult data might choose to log these messages or display them to the user.

<metadata>

The metadata element is optional, but may appear one or more times as a child of the Benchmark, Rule, Group, Value, Profile, TestResult, or rule-result elements. It contains document metadata expressed in XML. The default format for Benchmark document metadata is the Dublin Core Metadata Initiative (DCMI) Simple DC Element specification, as described in [12] and [14]. An example of the default format is shown in Example 11. Tools, especially document generation tools, should be prepared to process Dublin Core metadata in this element. This said, any XML metadata structures, including ad-hoc structures, can be included in a metadata field. Because any structures can appear in this field, authors are strongly encouraged to tag metadata with xmlns and, if available, schemaLocation attributes in order to identify the metadata structure utilized.

Content:	element
Cardinality:	0-n
Parent Elements:	Benchmark, Rule, Group, Value, Profile, TestResult, rule-result

Attributes:	*none*
Child Elements:	*non-XCCDF* (Dublin Core elements recommended)

Another suitable metadata format for XCCDF Benchmarks is the XML description format mandated by NIST for its Security Configuration Checklist Program [13].

Metadata is a powerful tool for authors to provide added value to their content. However, XCCDF puts some limits on the use of metadata. Specifically:

- Metadata may not duplicate the functionality of existing fields within XCCDF. If the information matches the common use of some other XCCDF field, the information must appear in that XCCDF field rather than as metadata.
- Metadata must not change the result of an automated assessment of the Benchmark. Tools should perform XCCDF assessments in the same way regardless of the presence of metadata. This said, metadata can still be used to support assessment features that are outside the purview of XCCDF. For example, using metadata to hold post-processing instructions to be performed on the completed XCCDF results is permitted.
- Metadata must not change what content is present when an XCCDF document is converted to prose form. This means that the same text must appear regardless of the presence of metadata. This said, metadata can contain instructions that might cause different stylistic conventions to be adopted in the conversion of an XCCDF document to prose.

Example 11 – Example of Benchmark Metadata Expressed with Dublin Core Elements

```
<cdf:metadata xmlns:dc="http://purl.org/dc/elements/1.1/">
   <dc:title>Security Benchmark for Ethernet Hubs</dc:title>
   <dc:creator>James Smith</dc:creator>
   <dc:publisher>Center for Internet Security</dc:publisher>
   <dc:subject>network security for layer 2 devices</dc:subject>
</cdf:metadata>
```

<model>

This element contains a specification for a suggested scoring model. This element may only appear as a child of a Benchmark and has one mandatory attribute: the URI of the scoring model. Some models may need additional parameters; to support such a model, zero or more ‘param’ elements may appear as children of the ‘model’ element.

Content:	element
Cardinality:	0-n
Parent Elements:	Benchmark
Attributes:	**system**
Child Elements:	param

This specification defines the following three scoring model URIs; compliance checking tool developers may define additional models. For more information see Section 3.3.

- `urn:xccdf:scoring:default` – this is the default weighted aggregated model. All tools must support this model and it is the default for Benchmarks that do not include any other model specifications.
- `urn:xccdf:scoring:flat` – this model computes the sum of the weights of rules that pass. All tools should support this model.
- `urn:xccdf:scoring:flat-unweighted` – this simple model simply computes the number of rules that passed. It does not use weights. All tools should support this model.
- `urn:xccdf:scoring:absolute` – this extremely simple model gives a score of 1 if all applicable rules passed and 0 otherwise.

<new-result>

An override rule result status. This element appears in an override element, inside a rule-result element, in a TestResult object. Its content must be one of the result status values listed in Section 3.2.

Content:	string
Cardinality:	1
Parent Elements:	override
Attributes:	*none*
Child Elements:	*none*

<notice>

This string element may only appear as the child of a Benchmark element, and supplies legal notice or copyright text about the Benchmark document. The id attribute must be a unique identifier.

Content:	mixed
Cardinality:	0-n
Parent Elements:	Benchmark
Attributes:	id
Child Elements:	*xhtml elements*

The notice element may contain XHTML markup to give it internal structure and formatting.

<old-result>

This element holds an overridden rule result status. This element appears in an override element, inside a rule-result element, in a TestResult object. Its content must be one of the result status values listed in Section 3.2.

Content:	string
Cardinality:	1
Parent Elements:	override
Attributes:	*none*
Child Elements:	*none*

<organization>

This element may appear only as a child of a TestResult. It contains the name of the organization, agency, department, or other entity responsible for the results.

Content:	string
Cardinality:	0-n
Parent Elements:	TestResult
Attributes:	*none*
Child Elements:	*none*

When multiple organization elements appear in a TestResult to indicate multiple organization names in a hierarchical organization, the highest-level organization should appear first (e.g., "U.S. Government") followed by subordinate organizations (e.g., "Department of Defense").

<override>

This element may appear only as a child of a rule-result and represents a human override of a Benchmark rule check result. It consists of five parts: a timestamp, the name of the human authority for the override, the old and new result status values, and remark text.

Content:	elements
Cardinality:	0-n
Parent Elements:	rule-result
Attributes:	**time**, **authority**
Child Elements:	**old-result**, **new-result**, **remark**

Example 12, below, shows how an override block would appear in a rule-result.

Example 12 – Example of rule-result Element with an Override

```
<cdf:rule-result idref="rule76"
          time="2005-04-26T14:38:19Z" severity="low">
  <cdf:result>pass</cdf:result>
  <cdf:override time="2005-04-26T15:03:20Z" authority="Bob Smith">
      <cdf:old-result>fail</cdf:old-result>
      <cdf:new-result>pass</cdf:new-result>
       <cdf:remark>
           Manual inspection showed this rule be satisfied. The
```

```
          relevant registry key was protected per policy, but with
          a more restrictive ACL than the benchmark was designed
          to check. The rule result has been overridden to 'pass'.
      </cdf:remark>
  </cdf:override>
  <cdf:instance context="registry">
       HKLM\SOFTWARE\Policies
  </cdf:instance>
  <cdf:check system="http://www.mitre.org/XMLSchema/oval">
      <cdf:check-content-ref href="oval-out.xml" name="OVAL123"/>
  </cdf:check>
</cdf:rule-result>
```

Note: if an override is added to a rule-result, then it will break any digital signature applied to the enclosing TestResult object.

<param>

This element may appear only as a child of a model element. It supplies a parameter that the compliance checking tool will need when computing the score using that model. None of the scoring models defined in the XCCDF specification require parameters, but proprietary models may.

Content:	string
Cardinality:	0-n
Parent Elements:	model
Attributes:	**name**
Child Elements:	*none*

Param elements with equal values for the name attribute may not appear as children of the same model element.

<plain-text>

This element holds a re-usable chunk of text for a Benchmark. It may be used anywhere that XHTML content or the XCCDF sub element may be used. Each plain-text definition must have a unique id.

Content:	string
Cardinality:	0-n
Parent Elements:	Benchmark,
Attributes:	**id**
Child Elements:	*none*

Note that plain-text definitions may only appear as children of Benchmark. The id on a plain-text definition must not be equal to any of the ids on Value, Group, or Rule objects.

<platform>

The 'platform' element specifies a target platform for the Benchmark, Rule, Group, Profile, or TestResult. This element has no content; the platform identifier appears in the 'idref' attribute. Multiple 'platform' elements may appear as children of a Benchmark, Rule, Group, or Profile, if the entity is suitable for multiple kinds of target systems.

Content:	*none*
Cardinality:	0-n
Parent Elements:	Benchmark, Group, Rule, Profile, TestResult
Attributes:	**idref**, override (Group, Rule, or Profile only)
Child Elements:	*none*

The platform element is optional. It indicates a platform with a CPE 2.3 Name or a CPE 2.3 platform specification identifier. The name or identifier appears in the idref attribute.

The override attribute may not appear when the platform element is a child of a Benchmark or TestResult.

<platform-specification>

The platform-specification element defines a list of identifiers for compound platform definitions written in the CPE 2.3 Language [17]. The identifiers defined in the platform-specification element may be used anywhere in the Benchmark in place of a CPE Name.

Content:	elements *(from the CPE 2.3 language namespace)*
Cardinality:	0-1
Parent Elements:	Benchmark
Attributes:	*none*
Child Elements:	*special*

<platform-definitions>, <Platform-Specification>

Each of these elements contains information about platforms to which the Benchmark may apply. The <platform-definitions> element contains a set of platform component and platform definitions using the CIS platform schema; it is permitted for compatibility with XCCDF 1.0. Both of these elements are deprecated, and appear in this specification only for backward compatibility. Common Platform Enumeration (CPE) Names should be used instead, see page 66.

Content:	elements
Cardinality:	0-1

Parent Elements:	Benchmark
Attributes:	*none*
Child Elements:	*special*

<profile>

This element specifies the Benchmark Profile used in applying a Benchmark; it can appear only as a child of a TestResult element.

Content:	*none*
Cardinality:	0-1
Parent Elements:	TestResult
Attributes:	**idref**
Child Elements:	*none*

<profile-note>

This element holds descriptive text about how one or more Profiles affect a Rule. It can appear only as a child of the Rule element.

Content:	mixed
Cardinality:	0-n
Parent Elements:	Rule
Attributes:	**tag**, xml:lang
Child Elements:	sub, *xhtml elements*

The mandatory ‘tag’ attribute holds an identifier; Profiles can refer to this identifier using the ‘note-tag’ attribute.

<question>

This element specifies an interrogatory string with which to prompt the user during tailoring. It may also be included into a generated document. Note that this element may not contain any XCCDF child elements, nor may it contain XHTML formatting elements. Multiple instances may appear with different xml:lang attributes.

Content:	string
Cardinality:	0-n
Parent Elements:	Group, Rule, Value
Attributes:	xml:lang, override
Child Elements:	*none*

For Rule and Group objects, the question text should be a simple binary (yes/no) question, because tailoring for Rules and Groups is for selection only. For Value objects, the question should reflect the designed data value needed for tailoring.

<rationale>

This element, which may appear as a child of a Group or Rule element, provides text that explains why that Group or Rule is important to the security of a target platform.

Content:	mixed
Cardinality:	0-n
Parent Elements:	Group, Rule
Attributes:	xml:lang, override
Child Elements:	sub, *xhtml elements*

<rear-matter>

This element contains textual content intended for use during Document Generation processing only; it is concluding material that should appear at or near the end of the generated document. Multiple instances may appear with different xml:lang attributes.

Content:	mixed
Cardinality:	0-n
Parent Elements:	Benchmark
Attributes:	xml:lang
Child Elements:	sub, *xhtml elements*

<reference>

This element provides supplementary descriptive text for a Benchmark, Rule, Group, or Value. It may have a simple string value or a value consisting of simple Dublin Core elements as described in [14]. It may also have an attribute, 'href', giving a URL for the referenced resource. Multiple reference elements may appear; a document generation processing tool may concatenate them, or put them into a reference list, and may choose to number them.

Content:	string or elements
Cardinality:	0-n
Parent Elements:	Benchmark, Group, Rule, Value, Profile
Attributes:	href
Child Elements:	*none or Dublin Core Elements*

References should be given as Dublin Core descriptions; a bare string is allowed for simplicity. If a bare string appears, then it is taken to be the string content for a Dublin Core 'title' element. For more information, consult [12].

<refine-rule>

This element adjusts the check tailoring selector, weight, and severity properties of a Rule. It has five attributes: an idref to a Rule, Group, or Item cluster (mandatory), and adjusted values for check selector, weight, severity, and role (all optional). Tailoring of the role property is now deprecated.

Content:	*none*
Cardinality:	0-n
Parent Elements:	Profile
Attributes:	**idref**, role ***(deprecated)***, selector, severity, weight
Child Elements:	*none*

The 'idref' attribute must match the id attribute of a Rule, Group, or a cluster-id of one or more Items in the Benchmark. When the 'idref' attribute refers to a Group, or if some of the Items in a cluster are Groups, then only the 'weight' attribute applies to them.

Selector tags apply only to the check children of a Rule. If the selector tag specified in a refine-rule element in a Profile does not match any of the selectors specified on any of the check children of a Rule, then the check child element without a selector attribute must be used.

<refine-value>

This element specifies the selector tag to be applied when tailoring a Value during use of a particular Profile. It has three attributes: the id of a Value or Item cluster (mandatory), the id of a Value selector tag, and a new setting for the Value operator.

Content:	*none*
Cardinality:	0-n
Parent Elements:	Profile
Attributes:	**idref**, operator, selector
Child Elements:	*none*

The 'idref' attribute must match the id attribute of a Value or a cluster-id of one or more Items in the Benchmark

The 'operator' attribute of the refine-value element can be used to override or replace the operator attribute of a Value. This is useful for refining the semantics of a Value.

Selector tags apply to the following child elements of Value: choices, default, complex-default, value, complex-value, match, lower-bound, and upper-bound. If the selector tag specified in a refine-value element in a Profile does not match any of the selectors specified on any of the Value children, then the child with no selector tag is used. If there is no child element of a given type without a selector (e.g., no default children without a selector), then the Value is processed as if there are no children of that particular type (e.g. process the Value as if no default information is provided). The exception to this are the value and complex-value children since a Value requires at least one instance of one of these fields to be present. If a given selector would normally mean that there were no effective value or complex-value fields, the first value or

complex-value child would become effective, even if it did not otherwise match the specified selector. The example below illustrates how selector tags and the refine-value element work.

Example 13 – Example of Profile refine-value Selector Tags

```
<cdf:Value id="pw-length" type="number" operator="equals">
   <cdf:title>Minimum password length policy</cdf:title>
   <cdf:value>8</cdf:value>
   <cdf:value selector="high">14</cdf:value>
   <cdf:lower-bound>8</cdf:lower-bound>
   <cdf:lower-bound selector="high">12</cdf:lower-bound>
</cdf:Value>
<cdf:Profile id="enterprise-internet">
     <cdf:title>Enterprise internet server profile</cdf:title>
     <cdf:refine-value idref="pw-length" selector="high"/>
</cdf:Profile>
```

<remark>

The remark element may appear as the child of a TestResult or override element; it contains a textual remark about the test.

Content:	string
Cardinality:	0-n (TestResult), 1 (override)
Parent Elements:	TestResult, override, refine-rule, refine-value, select
Attributes:	xml:lang
Child Elements:	*none*

The remark content may not contain any XHTML tags or other structure; it must be a plain string.

<requires>

The requires element may be a child of any Group or Rule and it specifies the id properties of one or more other Group or Rule, at least one of which must be selected in order for this Item to be selected. In a sense, the requires element is the opposite of the conflicts element. Each requires element specifies a list of one or more required Items by their ids, using the 'idref' attribute. If more than one id is given, then if at least one of the specified Groups or Rules is selected, then the requirement is met.

Content:	*none*
Cardinality:	0-n
Parent Elements:	Group, Rule
Attributes:	**idref**
Child Elements:	*none*

<result>

This simple element holds the verdict of applying a Benchmark Rule to a target or component of a target. It may have only one of nine values: “pass”, “fail”, “error”, “unknown”, “notchecked”, “notapplicable”, “notselected”, “fixed” or “informational”. For more information see page 34.

Content:	string
Cardinality:	1
Parent Elements:	rule-result
Attributes:	*none*
Child Elements:	*none*

<rule-result>

This element holds the result of applying a Rule from the Benchmark to a target system or component of a target system. It may only appear as the child of a TestResult element.

Content:	elements
Cardinality:	0-n
Parent Elements:	TestResult
Attributes:	**idref**, role ***(deprecated)***, time, severity, version, weight
Child Elements:	**result**, override, ident, metadata, message, instance, fix, check

The ‘idref’ attribute of a rule-result element must refer to a Rule element in the Benchmark. The result child element expresses the result (pass, fail, error, etc.) of applying the Rule to the target system. If the Rule is multiply instantiated, the instance elements indicate the particular system component and/or the check elements indicate the particular check relevant to a given result. If present, the override element provides information about a human override of a computed result status value.

<score>

This element contains the weighted score for a Benchmark test as a real number. Scoring models are defined in Section 3.3. This element may only appear as a child of a TestResult element.

Content:	decimal (non-negative number)
Cardinality:	1-n
Parent Elements:	TestResult
Attributes:	system, maximum
Child Elements:	*none*

The system attribute, a URI, identifies the scoring model (see the description of the model element on page 51 for a list of pre-defined models). If the system attribute does not appear,

then the model used was the default model. The maximum attribute, a real number, gives the maximum possible value of the score for this Benchmark test. If the maximum attribute does not appear, then it is taken to have a value of 100.

<select>

This element is part of a Profile; it overrides the selected attribute of a Rule or Group. Two attributes must be given with this element: the id of a Rule or Group (idref), and a boolean value (selected). If the boolean value is given as true, then the Rule or Group is selected for this Profile, otherwise it is unselected for this Profile.

Content:	*none*
Cardinality:	0-n
Parent Elements:	Profile
Attributes:	**idref**, **selected**
Child Elements:	*none*

The 'idref' attribute must match the id attribute of a Group or Rule in the Benchmark, or the cluster id assigned to one or more Rules or Groups.

<set-complex-value>

This element specifies a complex value for a Value object. It may appear as part of a Profile; in that case it overrides the value and complex-value properties of a Value object. It may appear as part of a TestResult; in that case it supplies the complex value used in the test. This element has one mandatory attribute. At least one of item (for a list) or external-type (for one or more instances of an externally defined data type) must appear as a child of this element.

Content:	string
Cardinality:	0-n
Parent Elements:	Profile, TestResult
Attributes:	**idref**
Child Elements:	item, external-type

In the content of a Profile, the identifier given for the 'idref' attribute may be a cluster id, in which case it applies only to the Value item members of the cluster; in the context of a TestResult, the identifier must match the id of a Value object in the Benchmark

<set-value>

This element specifies a value for a Value object. It may appear as part of a Profile; in that case it overrides the value or complex-value properties of a Value object. It may appear as part of a TestResult; in that case it supplies the value used in the test. This element has one mandatory attribute and no child elements.

Content:	string
Cardinality:	0-n
Parent Elements:	Profile, TestResult
Attributes:	**idref**
Child Elements:	*none*

In the content of a Profile, the identifier given for the 'idref' attribute may be a cluster id, in which case it applies only to the Value item members of the cluster; in the context of a TestResult, the identifier must match the id of a Value object in the Benchmark

<signature>

This element can hold an enveloped digital signature expressed according to the XML Digital Signature standard [10]. This element takes no attributes, and must contain exactly one element from the XML-Signature namespace.

Content:	elements
Cardinality:	0-1
Parent Elements:	Benchmark, Rule, Group, Value, Profile, TestResult
Attributes:	*none*
Child Elements	*Signature* (in XML-Signature namespace)

At most one enveloped signature can appear in an XCCDF document. If multiple signatures are needed, others must be detached signatures.

The signature element should contain one child element, Signature from the XML-Signature namespace, and that Signature should contain exactly one Reference to the block enclosing the signature. The Reference URI should be a relative URI to the Id attribute value of the enclosing object. For example, if the Id="abc", then the Reference URI would be "#abc".

<source>

The source element contains a URI indicating where a tailoring or benchmarking tool might obtain the value, or information about the value, for a Value object. XCCDF does not attach any meaning to the URI; it may be an arbitrary community or tool-specific value, or a pointer directly to a resource.

Content:	*none*
Cardinality:	0-n
Parent Elements:	Value
Attributes:	uri
Child Elements:	*none*

If several values for the source property appear, then they represent alternative means or locations for obtaining the value in descending order of preference (i.e., most preferred first).

<status>

This element provides a revision or standardization status for a Benchmark, along with the date at which the Benchmark attained that status. It must appear at least once in a Benchmark object, and may appear in any Item. If an Item does not have its own status element, its status is that of its parent element. The permitted string values for status are “accepted”, “deprecated”, “draft”, “interim”, and “incomplete”.

Content:	string (enumerated choices)
Cardinality:	0-n
Parent Elements:	Benchmark, Rule, Group, Value, Profile
Attributes:	date
Child Elements:	*none*

A Benchmark must have at least one status child element.

<sub>

This element represents a reference to a parameter value that may be set during tailoring. The element never has any content and must have its single attribute, value. The value attribute must equal the id attribute of a Value object or plain-text definition.

Content:	*none*
Cardinality:	0-n
Parent Elements:	description, fix, fixtext, front-matter, rationale, rear-matter, profileNote, title, warning
Attributes:	**idref**
Child Elements:	*none*

<target>

This element gives the name or description of a target system to which a Benchmark test was applied. It may only appear as a child of a TestResult element.

Content:	string
Cardinality:	1-n
Parent Elements:	TestResult
Attributes:	*none*
Child Elements:	*none*

<target-address>

This element gives the network address of a target system to which a Benchmark test was applied. It may only appear as a child of a TestResult element.

Content:	string
Cardinality:	0-n
Parent Elements:	TestResult
Attributes:	*none*
Child Elements:	*none*

<target-facts>

The TestResult object must be able to hold an arbitrary set of facts about the target of a test. This element holds those facts, each one of which is a fact element. It is an optional member of TestResult.

Content:	string
Cardinality:	0-1
Parent Elements:	TestResult
Attributes:	*none*
Child Elements:	fact

<title>

This element provides the descriptive title for a Benchmark, Rule, Group, Value, Profile, or TestResult. Multiple instances may appear with different languages (different values of the xml:lang attribute).

Content:	string
Cardinality:	0-n
Parent Elements:	Benchmark, Value, Group, Rule, Profile, TestResult
Attributes:	xml:lang, override
Child Elements:	*none*

This element may not contain XHTML markup.

The ‘override’ attribute controls how the title property is inherited, if the Item in which it appears extends another Item.

<upper-bound>

This element may appear zero or more times as a child of a Value element. It is used to constrain value input during tailoring, when the Value’s type is “number”. It contains a number; values supplied by the user for tailoring the Benchmark must be no greater than this number.

This element may have a selector tag attribute, which identifies it for Value refinement by a Profile. If more than one upper-bound element appears as the child of a Value, at most one may omit the selector attribute.

Content:	number
Cardinality:	0-n
Parent Elements:	Value
Attributes:	selector
Child Elements:	*none*

<value>

This string element is used to hold the value of a Value object. It or a complex-value element must appear as the child of a Value element, and no child elements. This element may have a selector tag attribute, which identifies it for Value refinement by a Profile. This element may appear more than once, but at most one of the sibling instances of this element or sibling complex-value elements may omit the selector tag.

Content:	String
Cardinality:	0-n (at least one of value or complex-value must be present in each Value)
Parent Elements:	Value
Attributes:	selector
Child Elements:	*None*

<version>

This element gives a version number for a Benchmark, Group, Rule, Value, or Profile. The version number content may be any string. This element allows an optional time attribute, which is a timestamp of when the object was defined. This element also allows an optional update attribute, which should be the URI specifying where updates to the object may be obtained.

Content:	string
Cardinality:	1 (Benchmark), 0-1 (all others)
Parent Elements:	Benchmark, Group, Rule, Value, Profile
Attributes:	time, update
Child Elements:	*none*

<warning>

This element provides supplementary descriptive text for a Benchmark, Rule, Group, or Value. The optional 'category' attribute may be used to provide a hint as to the nature of the warning.

Multiple warning elements may appear; processing tools should concatenate them for generating reports or documents (see also next section).

Content:	mixed
Cardinality:	0-n
Parent Elements:	Benchmark, Group, Rule, Value
Attributes:	category, xml:lang, override
Child Elements:	sub, *xhtml elements*

This element is intended to convey important cautionary information for the Benchmark user (e.g., "Complying with this rule will cause the system to reject all IP packets"). Processing tools may present this information in a special manner in generated documents.

If used, the category attribute must have one of the following values:

Value	Meaning
general	Broad or general-purpose warning (default for compatibility for XCCDF 1.0)
functionality	Warning about possible impacts to functionality or operational features
performance	Warning about changes to target system performance or throughput
hardware	Warning about hardware restrictions or possible impacts to hardware
legal	Warning about legal implications
regulatory	Warning about regulatory obligations or compliance implications
management	Warning about impacts to the management or administration of the target system
audit	Warning about impacts to audit or logging
dependency	Warning about dependencies between this Rule and other parts of the target system, or version dependencies

4.3. Handling Text and String Content

This sub-section provides additional information about how XCCDF processing tools must handle textual content in Benchmarks.

XHTML Formatting and Locale

Some text-valued XCCDF elements may contain formatting specified with elements from the XHTML Core Recommendation.

Many of the string and textual elements of XCCDF are listed as appearing multiple times under the same parent element. These elements, listed below, can have an xml:lang attribute that specifies the natural language locale for which they are written (e.g., "en" for English, "fr" for French). A processing tool should employ these attributes when possible during tailoring, document generation, and producing compliance reports, to create localized output. An example of using the xml:lang attribute is shown below.

Example 14 – A Simple Value Object with Questions in Different Languages

```
<cdf:Value id="web-server-port" type="number">
  <cdf:title>Web Server Port</cdf:title>
  <cdf:question xml:lang="en">
     What is the web server's TCP port?
  </cdf:question>
  <cdf:question xml:lang="fr">
     Quel est le port du TCP du web serveur?
  </cdf:question>
  <cdf:value>80</cdf:value>
</cdf:Value>
```

Multiple values for the same property in a single Item are handled differently, as described below. Multiple instances with different values of their xml:lang attribute are always permitted; an Item with no value for the xml:lang attribute is taken to have the same language as the Benchmark itself (as given by the xml:lang attribute on the Benchmark element).

Elements	Inheritance Behavior
description, title, fixtext, rationale, question, front-matter, rear-matter	At most one instance per language; inherited values with the same language get replaced
warning, reference, notice	Multiple instances treated as an ordered list; inherited instances prepended to the list

The platform element may also appear multiple times, each with a different id, to express the notion that a Rule, Group, Benchmark, or Profile applies to several different products or systems.

String Substitution and Reference Processing

There are three kinds of string substitution and one kind of reference processing that XCCDF document generation and reporting tools must support.

1. XCCDF **sub** element -
   The sub element supports substitution of information from a Value object, or the string content of a plain-text definition. The formatting for a sub element reference to a Value object is implementation-dependent for document generation, as described in Section 3.3. Formatting for a sub element reference to a plain-text definition is very simple: the string content of the plain-text definition replaces the sub element.
2. XHTML **object** element -
   The object element supports substitutions of a variety of information from another Item or Profile, or the string content of a plain-text definition. To avoid possible conflicts with uses of an XHTML object that should not be processed specially, all XCCDF object references must be a relative URI beginning with "`#xccdf:`". The following URI values can be used to refer to things from an "object" element, using the "data" attribute:
   - **`#xccdf:value:`*`id`***
     Insert the value of the plain-text block, Value, or fact with id *id*:. When a URI of this form is used, the value of the reference should be substituted for the entire "object"

element and its content (if any). In keeping with the standard semantics of XHTML, if the *id* cannot be resolved, then the textual content of the "object" element should be retained.

- **`#xccdf:title:`*`id`***

  Insert the string content of the "title" child of the Item with id *id*. Use the current language value locale setting, if any. When a URI of this form is used, the title string should be substituted for the entire "object" element and its content (if any). In keeping with the standard semantics of XHTML, if the *id* cannot be resolved, then the textual content of the "object" element should be retained.

3. XHTML anchor (**a**) element -
   The anchor element can be used to create an intra-document link to another XCCDF Item or Profile. To avoid possible conflicts with uses of the XHTML anchor element that should not be processed specially, all XCCDF anchor references must be a relative URI beginning with "`#xccdf:`". The following URI values can be used to refer to things from an anchor ("a") element, using the 'href' attribute:

   - **`#xccdf:link:`*`id`***

     Create an intra-document link to the point in the document where the Item *id* is described. The content of the element should be the text of the link.

5. Conclusions

The XCCDF specification defines a means for expressing security guidance documents in a way that should foster development of interoperable tools and content. It is designed to permit the same document to serve in several roles:

- Source code for generation of publication documents and hardcopy
- Script for eliciting local security policy settings and values from a user
- Structure for containing and organizing code that drives system analysis and configuration checking engines
- Source code for text to appear in security policy compliance reports
- A record of a compliance test, including the results of applying various rules
- Structure for expressing compliance scoring/weighting decisions.

XCCDF 1.2 was designed as a compatible extension of 1.0 and 1.1, based on suggestions from early adopters and potential users. Many features have been added, but every valid 1.0 and 1.1 document should be a valid 1.2 document, once the namespace is adjusted. The forward compatibility will help ensure that content written for XCCDF 1.0 will not be made obsolete by the 1.2 specification.

Adoption of a common format should permit security professionals, security tool vendors, and system auditors to exchange information more quickly and precisely, and also permit greater automation of security testing and configuration checking.

Appendix A. XCCDF Schema

The XML Schema below describes XCCDF in a manner that should allow automatic validation of most aspects of the format. It is not possible to express all of the constraints that XCCDF imposes in a Schema, unfortunately, and a few of the constraints that it is possible to express have been omitted for simplicity.

Whether or not to validate XCCDF XML documents is an implementation decision left to tool developers, but it is strongly recommended.

XCCDF 1.2 Schema

```
<?xml version="1.0" encoding="UTF-8"?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:cdf="http://checklists.nist.gov/xccdf/1.2"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:cisp="http://www.cisecurity.org/xccdf/platform/0.2.3"
    xmlns:cdfp="http://checklists.nist.gov/xccdf-p/1.1"
    xmlns:cpe1="http://cpe.mitre.org/XMLSchema/cpe/1.0"
    xmlns:cpe2="http://cpe.mitre.org/language/2.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-Instance"
    targetNamespace="http://checklists.nist.gov/xccdf/1.2"
    elementFormDefault="qualified" attributeFormDefault="unqualified">

    <xsd:annotation>
        <xsd:documentation xml:lang="en"> This schema defines the eXtensible
            Configuration Checklist Description Format (XCCDF), a data format
            for defining security benchmarks and checklists, and for recording
            the results of applying such benchmarks. For more information,
            consult the specification document, "Specification for the
            Extensible Configuration Checklist Description Format", version 1.2.
            This schema was developed by Neal Ziring, with ideas and assistance
            from David Waltermire. The following helpful individuals also
            contributed ideas to the definition of this schema: David Proulx,
            Andrew Buttner, Ryan Wilson, Matthew Kerr, Stephen Quinn. Ian
            Crawford found numerous discrepancies between this schema and the
            spec document. Peter Mell and his colleagues also made many
            suggestions. <version date="1 June 2010">1.2</version>
        </xsd:documentation>
        <xsd:appinfo>
            <schema>XCCDF Language</schema>
            <author>Neal Ziring</author>
            <version>1.2</version>
            <date>2010-06-01</date>
        </xsd:appinfo>
    </xsd:annotation>

    <!-- Import base XML namespace -->
    <xsd:import namespace="http://www.w3.org/XML/1998/namespace"
        schemaLocation="xml.xsd">
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> Import the XML namespace because
                this schema uses the xml:lang and xml:base attributes.
            </xsd:documentation>
        </xsd:annotation>
    </xsd:import>

    <!-- Import Dublin Core metadata namespace -->
    <xsd:import namespace="http://purl.org/dc/elements/1.1/"
        schemaLocation="simpledc20021212.xsd">
```

```
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> Import the simple Dublin Core
                namespace because this schema uses it for benchmark metadata and
                for references. </xsd:documentation>
        </xsd:annotation>
    </xsd:import>

    <!-- Import CIS platform specification namespace - DEPRECATED -->
    <xsd:import namespace="http://www.cisecurity.org/xccdf/platform/0.2.3"
        schemaLocation="platform-0.2.3.xsd">
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> Import the CIS platform schema,
                which we use for describing target IT platforms in the
                Benchmark. The CIS platform schema was designed by David
                Waltermire. Use of the CIS platform schema in XCCDF benchmarks
                is deprecated. The CIS platform schema is included only for
                backward compatibility with XCCDF 1.0. Use CPE 2 instead.
            </xsd:documentation>
        </xsd:annotation>
    </xsd:import>

    <!-- Import XCCDF-P platform specification namespace - DEPRECATED -->
    <xsd:import namespace="http://checklists.nist.gov/xccdf-p/1.1"
        schemaLocation="xccdfp-1.1.xsd">
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> Import the XCCDF-P platform
                schema, which we use for describing target IT platforms in the
                Benchmark. The CIS platform schema was designed by Neal Ziring
                using ideas and concepts developed by DISA, CIS, and others. Use
                of XCCDF-P platform specification in XCCDF benchmarks is
                deprecated. XCCDF-P is included in this schema only for backward
                compatibility with version 1.1 and 1.1.2. Use CPE 2 instead.
            </xsd:documentation>
        </xsd:annotation>
    </xsd:import>

    <!-- Import CPE 1.0 dictionary namespace - DEPRECATED -->
    <xsd:import namespace="http://cpe.mitre.org/XMLSchema/cpe/1.0"
        schemaLocation="cpe-1.0.xsd">
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> Import the Common Platform
                Enumeration XML schema, which can be used for naming and
                describing target IT platforms in the Benchmark. Every CPE name
                is a URI that begins with "cpe:". For more details see "Common
                Platform Enumeration (CPE) - Specification", by Buttner,
                Wittbold, and Ziring (2007). Use of CPE 1.0 definitions in XCCDF
                benchmarks is deprecated. CPE 1.0 is included in this schema
                only for backward compatibility with XCCDF 1.1.2 and 1.1.3. CPE
                2 Names should be used for simple (unitary) platforms, and CPE 2
                Language definitions used for complex platforms.
            </xsd:documentation>
        </xsd:annotation>
    </xsd:import>

    <!-- Import CPE 2.3 Language namespace -->
    <xsd:import namespace="http://cpe.mitre.org/language/2.0"
        schemaLocation="cpe-language_2.3.xsd">
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> Import the Common Platform
                Enumeration language schema, which can be used for defining
                compound CPE tests for complex IT platforms in the Benchmark.
                For more info see "Common Platform Enumeration (CPE) -
                Specification", Version 2.0" by Buttner and Ziring (Sept 2007).
            </xsd:documentation>
        </xsd:annotation>
```

```
    </xsd:import>

    <!-- ************************************************************ -->
    <!-- *****************  Benchmark Element  ********************** -->
    <!-- ************************************************************ -->
    <xsd:element name="Benchmark">
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> The benchmark tag is the top level
                element representing a complete security checklist, including
                descriptive text, metadata, test items, and test results. A
                Benchmark may be signed with a XML-Signature.
            </xsd:documentation>
        </xsd:annotation>
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="cdf:status" minOccurs="1"
                    maxOccurs="unbounded"/>
                <xsd:element name="dc-status" minOccurs="0" maxOccurs="1"
                    type="cdf:dc-statusType"/>
                <xsd:element name="title" type="cdf:textType" minOccurs="0"
                    maxOccurs="unbounded"/>
                <xsd:element name="description" type="cdf:htmlTextWithSubType"
                    minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element name="notice" type="cdf:noticeType" minOccurs="0"
                    maxOccurs="unbounded"/>
                <xsd:element name="front-matter" type="cdf:htmlTextWithSubType"
                    minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element name="rear-matter" type="cdf:htmlTextWithSubType"
                    minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element name="reference" type="cdf:referenceType"
                    minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element name="plain-text" type="cdf:plainTextType"
                    minOccurs="0" maxOccurs="unbounded"/>
                <!-- Choice of current (CPE 2.0) or deprecated platform
                     identifier elements. -->
                <xsd:choice minOccurs="0" maxOccurs="1">
                    <!-- CIS Platform schema, compatibility with XCCDF 1.0 -->
                    <xsd:element ref="cisp:platform-definitions" minOccurs="0"
                        maxOccurs="1"/>
                    <!-- XCCDF-P 1.0 schema, compatibility with XCCDF 1.1 -->
                    <xsd:element ref="cdfp:Platform-Specification" minOccurs="0"
                        maxOccurs="1"/>
                    <!-- CPE 1.0 schema, compatibility with XCCDF 1.1.3 -->
                    <xsd:element ref="cpe1:cpe-list" minOccurs="0" maxOccurs="1"/>
                    <!-- CPE 2.0 language schema, for SCAP 1.0 conformance -->
                    <xsd:element ref="cpe2:platform-specification" minOccurs="0"
                        maxOccurs="1"/>
                </xsd:choice>
                <xsd:element name="platform" type="cdf:CPE2idrefType"
                    minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element name="version" type="cdf:versionType" minOccurs="1"
                    maxOccurs="1"/>
                <xsd:element name="metadata" type="cdf:metadataType"
                    minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element ref="cdf:model" minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element ref="cdf:Profile" minOccurs="0"
                    maxOccurs="unbounded"/>
                <xsd:element ref="cdf:Value" minOccurs="0" maxOccurs="unbounded"/>
                <xsd:choice minOccurs="0" maxOccurs="unbounded">
                    <xsd:element ref="cdf:Group"/>
                    <xsd:element ref="cdf:Rule"/>
                </xsd:choice>
                <xsd:element ref="cdf:TestResult" minOccurs="0"
                    maxOccurs="unbounded"/>
                <xsd:element name="signature" type="cdf:signatureType"
```

```
                minOccurs="0" maxOccurs="1"/>
            </xsd:sequence>
            <xsd:attribute name="id" type="xsd:NCName" use="required"/>
            <!-- the 'Id' attribute is needed for XML-Signature -->
            <xsd:attribute name="Id" type="xsd:ID" use="optional"/>
            <xsd:attribute name="resolved" type="xsd:boolean" default="false"
                use="optional"/>
            <xsd:attribute name="style" type="xsd:string" use="optional"/>
            <xsd:attribute name="style-href" type="xsd:anyURI" use="optional"/>
            <xsd:attribute ref="xml:lang"/>
        </xsd:complexType>

        <xsd:unique name="noticeIdUnique">
            <xsd:annotation>
                <xsd:documentation xml:lang="en"> Legal notices must have unique
                    id values. </xsd:documentation>
            </xsd:annotation>
            <xsd:selector xpath="cdf:notice"/>
            <xsd:field xpath="@id"/>
        </xsd:unique>

        <xsd:key name="itemIdKey">
            <xsd:annotation>
                <xsd:documentation xml:lang="en"> Items must have unique id
                    values, and also they must not collide </xsd:documentation>
            </xsd:annotation>
            <xsd:selector
                xpath=".//cdf:Value|.//cdf:Group|.//cdf:Rule|./cdf:plain-text"/>
            <xsd:field xpath="@id"/>
        </xsd:key>

        <xsd:key name="modelSystemKey">
            <xsd:annotation>
                <xsd:documentation xml:lang="en"> Model system attributes must
                    be unique. </xsd:documentation>
            </xsd:annotation>
            <xsd:selector xpath="./cdf:model"/>
            <xsd:field xpath="@system"/>
        </xsd:key>

        <xsd:key name="valueIdKey">
            <xsd:annotation>
                <xsd:documentation xml:lang="en"> Value item ids are special
                    keys, need this for the valueIdKeyRef and valueExtIdKeyRef
                    keyrefs below. </xsd:documentation>
            </xsd:annotation>
            <xsd:selector xpath=".//cdf:Value"/>
            <xsd:field xpath="@id"/>
        </xsd:key>

        <xsd:key name="groupIdKey">
            <xsd:annotation>
                <xsd:documentation xml:lang="en"> Group item ids are special
                    keys, need this for the groupIdKeyRef keyref below.
                </xsd:documentation>
            </xsd:annotation>
            <xsd:selector xpath=".//cdf:Group"/>
            <xsd:field xpath="@id"/>
        </xsd:key>

        <xsd:key name="ruleIdKey">
            <xsd:annotation>
                <xsd:documentation xml:lang="en"> Rule items have a unique key,
                    we need this for the ruleIdKeyRef keyref below. (Rule key
                    refs are used by rule-results.) </xsd:documentation>
```

```
            </xsd:annotation>
            <xsd:selector xpath=".//cdf:Rule"/>
            <xsd:field xpath="@id"/>
        </xsd:key>

        <xsd:key name="selectableItemIdKey">
            <xsd:annotation>
                <xsd:documentation xml:lang="en"> Group and Rule item ids are
                    special keys, we need this for the requiresIdKeyRef keyref
                    below. </xsd:documentation>
            </xsd:annotation>
            <xsd:selector xpath=".//cdf:Group | .//cdf:Rule"/>
            <xsd:field xpath="@id"/>
        </xsd:key>

        <xsd:key name="plainTextValueIdKey">
            <xsd:annotation>
                <xsd:documentation xml:lang="en"> Plaintext objects and Value
                    objects each have and id, and they must be unique and not
                    overlap. </xsd:documentation>
            </xsd:annotation>
            <xsd:selector xpath="./cdf:plain-text | .//Value"/>
            <xsd:field xpath="@id"/>
        </xsd:key>

        <xsd:key name="profileIdKey">
            <xsd:annotation>
                <xsd:documentation xml:lang="en"> Profile objects have a unique
                    id, it is used for extension, too. </xsd:documentation>
            </xsd:annotation>
            <xsd:selector xpath="./cdf:Profile"/>
            <xsd:field xpath="@id"/>
        </xsd:key>

        <xsd:keyref name="valueExtIdKeyRef" refer="cdf:valueIdKey">
            <xsd:annotation>
                <xsd:documentation xml:lang="en"> An extends attribute on Value
                    object must reference an existing Value.
                </xsd:documentation>
            </xsd:annotation>
            <xsd:selector xpath=".//cdf:Value"/>
            <xsd:field xpath="@extends"/>
        </xsd:keyref>

        <xsd:keyref name="groupExtIdKeyRef" refer="cdf:groupIdKey">
            <xsd:annotation>
                <xsd:documentation xml:lang="en"> An extends attribute on Group
                    object must reference an existing Group.
                </xsd:documentation>
            </xsd:annotation>
            <xsd:selector xpath=".//cdf:Group"/>
            <xsd:field xpath="@extends"/>
        </xsd:keyref>

        <xsd:keyref name="ruleExtIdKeyRef" refer="cdf:ruleIdKey">
            <xsd:annotation>
                <xsd:documentation xml:lang="en"> An extends attribute on Rule
                    object must reference an existing Rule. </xsd:documentation>
            </xsd:annotation>
            <xsd:selector xpath=".//cdf:Rule"/>
            <xsd:field xpath="@extends"/>
        </xsd:keyref>

        <xsd:keyref name="profileExtIdKeyRef" refer="cdf:profileIdKey">
            <xsd:annotation>
```

```
                <xsd:documentation xml:lang="en"> An extends attribute on
                    Profile object must reference an existing Profile.
                </xsd:documentation>
            </xsd:annotation>
            <xsd:selector xpath="./cdf:Profile"/>
            <xsd:field xpath="@extends"/>
        </xsd:keyref>

        <xsd:keyref name="valueIdKeyRef" refer="cdf:valueIdKey">
            <xsd:annotation>
                <xsd:documentation xml:lang="en"> Check-export elements must
                    reference existing values. </xsd:documentation>
            </xsd:annotation>
            <xsd:selector xpath=".//cdf:check/cdf:check-export"/>
            <xsd:field xpath="@value-id"/>
        </xsd:keyref>

        <xsd:keyref name="subValueKeyRef" refer="cdf:plainTextValueIdKey">
            <xsd:annotation>
                <xsd:documentation xml:lang="en"> Sub elements must reference
                    existing Value or plain-text ids. </xsd:documentation>
            </xsd:annotation>
            <xsd:selector xpath=".//cdf:sub"/>
            <xsd:field xpath="@value"/>
        </xsd:keyref>

        <xsd:keyref name="ruleIdKeyRef" refer="cdf:ruleIdKey">
            <xsd:annotation>
                <xsd:documentation xml:lang="en"> The rule-result element idref
                    must refer to an existing Rule. </xsd:documentation>
            </xsd:annotation>
            <xsd:selector xpath="./cdf:TestResult/cdf:rule-result"/>
            <xsd:field xpath="@idref"/>
        </xsd:keyref>

        <xsd:keyref name="profileIdKeyRef" refer="cdf:profileIdKey">
            <xsd:annotation>
                <xsd:documentation xml:lang="en"> The requires a profile element
                    in a TestResult element to refer to an existing Profile
                </xsd:documentation>
            </xsd:annotation>
            <xsd:selector xpath="./cdf:TestResult/cdf:profile"/>
            <xsd:field xpath="@idref"/>
        </xsd:keyref>

    </xsd:element>

    <xsd:complexType name="noticeType" mixed="true">
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> Data type for legal notice element
                that has text content and a unique id attribute.
            </xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:any namespace="http://www.w3.org/1999/xhtml" minOccurs="0"
                maxOccurs="unbounded" processContents="skip"/>
        </xsd:sequence>
        <xsd:attribute name="id" type="xsd:NCName"/>
        <xsd:attribute ref="xml:base"/>
        <xsd:attribute ref="xml:lang"/>
    </xsd:complexType>

    <xsd:complexType name="dc-statusType">
        <xsd:annotation>
            <xsd:documentation> Type of the Dublin-Core status field
```

```
            </xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:any namespace="http://purl.org/dc/elements/1.1/" minOccurs="1"
                maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="plainTextType">
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> Data type for a reusable text
                block, with an unique id attribute. </xsd:documentation>
        </xsd:annotation>
        <xsd:simpleContent>
            <xsd:extension base="xsd:string">
                <xsd:attribute name="id" type="xsd:NCName" use="required"/>
            </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>

    <xsd:complexType name="referenceType" mixed="true">
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> Data type for a reference
                citation, an href URL attribute (optional), with content of text
                or simple Dublin Core elements. Elements of this type can also
                have an override attribute to help manage inheritance.
            </xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:any namespace="http://purl.org/dc/elements/1.1/"
                processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:attribute name="href" type="xsd:anyURI"/>
        <xsd:attribute name="override" type="xsd:boolean"/>
    </xsd:complexType>

    <xsd:complexType name="signatureType">
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> XML-Signature over the Benchmark;
                note that this will always be an 'enveloped' signature, so the
                single element child of this element should be dsig:Signature.
            </xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:any namespace="http://www.w3.org/2000/09/xmldsig#"
                processContents="skip" minOccurs="1" maxOccurs="1"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="metadataType">
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> Metadata for the Benchmark, should
                be Dublin Core or some other well-specified and accepted
                metadata format. If Dublin Core, then it will be a sequence of
                simple Dublin Core elements. The NIST checklist metadata should
                also be supported, although the specification document is still
                in draft in NIST special pub 800-70. </xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:any minOccurs="1" maxOccurs="unbounded" processContents="lax"
                namespace="##other"/>
        </xsd:sequence>
    </xsd:complexType>

    <!-- ********************************************************** -->
```

```
    <!-- *************  Global elements and types  ******************* -->
    <!-- ************************************************************ -->
    <xsd:element name="status">
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> The acceptance status of an Item
                with an optional date attribute that signifies the date of the
                status change. </xsd:documentation>
        </xsd:annotation>
        <xsd:complexType>
            <xsd:simpleContent>
                <xsd:extension base="cdf:statusType">
                    <xsd:attribute name="date" type="xsd:date" use="optional"/>
                </xsd:extension>
            </xsd:simpleContent>
        </xsd:complexType>
    </xsd:element>

    <xsd:element name="model">
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> A suggested scoring model for a
                Benchmark, also encapsulating any parameters needed by the
                model. Every model is designated with a URI, which appears here
                as the system attribute. </xsd:documentation>
        </xsd:annotation>
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="param" type="cdf:paramType" minOccurs="0"
                    maxOccurs="unbounded"/>
            </xsd:sequence>
            <xsd:attribute name="system" type="xsd:anyURI" use="required"/>
        </xsd:complexType>

        <xsd:key name="paramNameKey">
            <xsd:annotation>
                <xsd:documentation xml:lang="en"> Parameter names must be
                    unique. </xsd:documentation>
            </xsd:annotation>
            <xsd:selector xpath="./cdf:param"/>
            <xsd:field xpath="@name"/>
        </xsd:key>
    </xsd:element>

    <xsd:complexType name="paramType">
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> Type for a scoring model
                parameter: a name and a string value. </xsd:documentation>
        </xsd:annotation>
        <xsd:simpleContent>
            <xsd:extension base="xsd:string">
                <xsd:attribute name="name" type="xsd:NCName" use="required"/>
            </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>


    <xsd:simpleType name="statusType">
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> The possible status codes for an
                Benchmark or Item to be inherited from the parent element if it
                is not defined. </xsd:documentation>
        </xsd:annotation>
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="accepted"/>
            <xsd:enumeration value="deprecated"/>
            <xsd:enumeration value="draft"/>
```

```
            <xsd:enumeration value="incomplete"/>
            <xsd:enumeration value="interim"/>
        </xsd:restriction>
    </xsd:simpleType>

    <xsd:complexType name="versionType">
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> Type for a version number, with a
                timestamp attribute for when the version was made.
            </xsd:documentation>
        </xsd:annotation>
        <xsd:simpleContent>
            <xsd:extension base="xsd:string">
                <xsd:attribute name="time" type="xsd:dateTime" use="optional"/>
                <xsd:attribute name="update" type="xsd:anyURI" use="optional"/>
            </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>

    <!-- ************************************************************ -->
    <!-- ******************  Text Types  ************************** -->
    <!-- ************************************************************ -->
    <xsd:complexType name="textType">
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> Type for a string with an xml:lang
                attribute. Elements of this type can also have an override
                attribute to help manage inheritance. </xsd:documentation>
        </xsd:annotation>
        <xsd:simpleContent>
            <xsd:extension base="xsd:string">
                <xsd:attribute ref="xml:lang"/>
                <xsd:attribute name="override" type="xsd:boolean" use="optional"
                    default="0"/>
            </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>

    <xsd:complexType name="htmlTextType" mixed="true">
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> Type for a string with XHTML
                elements and xml:lang attribute. Elements of this type can also
                have an override attribute to help manage inheritance.
            </xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:any namespace="http://www.w3.org/1999/xhtml" minOccurs="0"
                maxOccurs="unbounded" processContents="skip"/>
        </xsd:sequence>
        <xsd:attribute ref="xml:lang"/>
        <xsd:attribute name="override" type="xsd:boolean" use="optional"
            default="0"/>
    </xsd:complexType>

    <xsd:complexType name="htmlTextWithSubType" mixed="true">
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> Type for a string with embedded
                Value substitutions and XHTML elements, and an xml:lang
                attribute. Elements of this type can also have an override
                attribute to help manage inheritance. [Note: this definition is
                rather loose, it allows anything whatsoever to occur insides
                XHTML tags inside here. Further, constraints of the XHTML schema
                do not get checked! It might be possible to solve this using XML
                Schema redefinition features.] </xsd:documentation>
        </xsd:annotation>
        <xsd:choice minOccurs="0" maxOccurs="unbounded">
```

```
            <xsd:element name="sub" type="cdf:idrefType"/>
            <xsd:any namespace="http://www.w3.org/1999/xhtml"
                processContents="skip"/>
        </xsd:choice>
        <xsd:attribute ref="xml:lang"/>
        <xsd:attribute name="override" type="xsd:boolean" use="optional"
            default="0"/>
    </xsd:complexType>

    <xsd:complexType name="profileNoteType" mixed="true">
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> Type for a string with embedded
                Value substitutions and XHTML elements, an xml:lang attribute,
                and a profile-note tag. </xsd:documentation>
        </xsd:annotation>
        <xsd:choice minOccurs="0" maxOccurs="unbounded">
            <xsd:element name="sub" type="cdf:idrefType"/>
            <xsd:any namespace="http://www.w3.org/1999/xhtml"
                processContents="skip"/>
        </xsd:choice>
        <xsd:attribute ref="xml:lang"/>
        <xsd:attribute name="tag" type="xsd:NCName" use="required"/>
    </xsd:complexType>

    <xsd:complexType name="textWithSubType" mixed="true">
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> Type for a string with embedded
                Value substitutions and an xml:lang attribute. Elements of this
                type can also have an override attribute to help manage
                inheritance. </xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:element name="sub" type="cdf:idrefType" minOccurs="0"
                maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:attribute ref="xml:lang"/>
        <xsd:attribute name="override" type="xsd:boolean" use="optional"
            default="0"/>
    </xsd:complexType>

    <xsd:complexType name="idrefType">
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> Data type for elements that have
                no content, just a mandatory id reference. </xsd:documentation>
        </xsd:annotation>
        <xsd:attribute name="idref" type="xsd:NCName" use="required"/>
    </xsd:complexType>

    <xsd:complexType name="idrefListType">
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> Data type for elements that have
                no content, just a space-separated list of id references.
            </xsd:documentation>
        </xsd:annotation>
        <xsd:attribute name="idref" type="xsd:NMTOKENS" use="required"/>
    </xsd:complexType>

    <xsd:complexType name="overrideableIdrefType">
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> Data type for elements that have
                no content, just a mandatory id reference, but also have an
                override attribute for controlling inheritance.
            </xsd:documentation>
        </xsd:annotation>
        <xsd:complexContent>
```

```
            <xsd:extension base="cdf:idrefType">
                <xsd:attribute name="override" type="xsd:boolean" use="optional"
                    default="0"/>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>

    <xsd:complexType name="CPE2idrefType">
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> Data type for elements that have
                no content, just a mandatory CPE 2 name as an id. (This is
                mainly for the platform element, which uses CPE 2 names and CPE
                Language identifers used as platform identifiers.)
            </xsd:documentation>
        </xsd:annotation>
        <xsd:attribute name="idref" type="xsd:string" use="required"/>
    </xsd:complexType>

    <xsd:complexType name="overrideableCPE2idrefType">
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> Data type for elements that have
                no content, just a mandatory CPE 2 reference, but also have an
                override attribute for controlling inheritance.
            </xsd:documentation>
        </xsd:annotation>
        <xsd:complexContent>
            <xsd:extension base="cdf:CPE2idrefType">
                <xsd:attribute name="override" type="xsd:boolean" use="optional"
                    default="0"/>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>


    <!-- ************************************************************* -->
    <!-- *************** Item Element (Base Class)  ***************** -->
    <!-- ************************************************************* -->
    <xsd:element name="Item" type="cdf:itemType">
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> Type element type imposes
                constraints shared by all Groups, Rules and Values. The itemType
                is abstract, so the element Item can never appear in a valid
                XCCDF document. </xsd:documentation>
        </xsd:annotation>
    </xsd:element>

    <xsd:complexType name="itemType" abstract="1">
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> This abstract item type represents
                the basic data shared by all Groups, Rules and Values
            </xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:element ref="cdf:status" minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element name="dc-status" minOccurs="0" maxOccurs="1"
                type="cdf:dc-statusType"/>
            <xsd:element name="version" type="cdf:versionType" minOccurs="0"
                maxOccurs="1"/>
            <xsd:element name="title" type="cdf:textWithSubType" minOccurs="0"
                maxOccurs="unbounded"/>
            <xsd:element name="description" type="cdf:htmlTextWithSubType"
                minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element name="warning" type="cdf:warningType" minOccurs="0"
                maxOccurs="unbounded"/>
            <xsd:element name="question" type="cdf:textType" minOccurs="0"
```

```
            maxOccurs="unbounded"/>
        <xsd:element name="reference" type="cdf:referenceType" minOccurs="0"
            maxOccurs="unbounded"/>
        <xsd:element name="metadata" type="cdf:metadataType" minOccurs="0"
            maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:NCName" use="required"/>
    <xsd:attribute name="abstract" type="xsd:boolean" default="false"
        use="optional"/>
    <xsd:attribute name="cluster-id" type="xsd:NCName" use="optional"/>
    <xsd:attribute name="extends" type="xsd:NCName" use="optional"/>
    <xsd:attribute name="hidden" type="xsd:boolean" default="false"
        use="optional"/>
    <xsd:attribute name="prohibitChanges" type="xsd:boolean" default="false"
        use="optional"/>
    <xsd:attribute ref="xml:lang"/>
    <xsd:attribute ref="xml:base"/>
    <xsd:attribute name="Id" type="xsd:ID" use="optional"/>
</xsd:complexType>


<!-- ************************************************************ -->
<!-- ************ Selectable Item Type (Base Class)  ************** -->
<!-- ************************************************************ -->
<xsd:complexType name="selectableItemType" abstract="true">
    <xsd:annotation>
        <xsd:documentation xml:lang="en"> This abstract item type represents
            the basic data shared by all Groups and Rules. It extends the
            itemType given above. </xsd:documentation>
    </xsd:annotation>
    <xsd:complexContent>
        <xsd:extension base="cdf:itemType">
            <xsd:sequence>
                <xsd:element name="rationale" type="cdf:htmlTextWithSubType"
                    minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element name="platform"
                    type="cdf:overrideableCPE2idrefType" minOccurs="0"
                    maxOccurs="unbounded"/>
                <xsd:element name="requires" type="cdf:idrefListType"
                    minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element name="conflicts" type="cdf:idrefType"
                    minOccurs="0" maxOccurs="unbounded"/>
            </xsd:sequence>
            <xsd:attribute name="selected" type="xsd:boolean" default="true"
                use="optional"/>
            <xsd:attribute name="weight" type="cdf:weightType" default="1.0"
                use="optional"/>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<!-- ************************************************************ -->
<!-- *********************  Group Element  ********************** -->
<!-- ************************************************************ -->
<xsd:element name="Group" type="cdf:groupType"/>

<xsd:complexType name="groupType">
    <xsd:annotation>
        <xsd:documentation xml:lang="en"> Data type for the Group element
            that represents a grouping of Groups, Rules and Values.
        </xsd:documentation>
    </xsd:annotation>
    <xsd:complexContent>
        <xsd:extension base="cdf:selectableItemType">
            <xsd:sequence>
```

```
                <xsd:element ref="cdf:Value" minOccurs="0"
                    maxOccurs="unbounded"/>
                <xsd:choice minOccurs="0" maxOccurs="unbounded">
                    <xsd:element ref="cdf:Group"/>
                    <xsd:element ref="cdf:Rule"/>
                </xsd:choice>
                <xsd:element name="signature" type="cdf:signatureType"
                    minOccurs="0" maxOccurs="1"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>


<!-- ************************************************************ -->
<!-- ********************  Rule Element  ************************** -->
<!-- ************************************************************ -->
<xsd:element name="Rule" type="cdf:ruleType">
    <xsd:annotation>
        <xsd:documentation xml:lang="en"> The Rule element contains the
            description for a single item of guidance or constraint. Rules
            form the basis for testing a target platform for compliance with
            a benchmark, for scoring, and for conveying descriptive prose,
            identifiers, references, and remediation information.
        </xsd:documentation>
    </xsd:annotation>
    <xsd:unique name="ruleCheckSelectorKey">
        <xsd:selector xpath="./cdf:check"/>
        <xsd:field xpath="@selector"/>
        <xsd:field xpath="@system"/>
    </xsd:unique>
    <xsd:unique name="ruleCheckIdKey">
        <xsd:selector xpath=".//cdf:check"/>
        <xsd:field xpath="@id"/>
    </xsd:unique>
</xsd:element>

<xsd:complexType name="ruleType">
    <xsd:annotation>
        <xsd:documentation xml:lang="en"> Data type for the Rule element
            that represents a specific benchmark test. </xsd:documentation>
    </xsd:annotation>
    <xsd:complexContent>
        <xsd:extension base="cdf:selectableItemType">
            <xsd:sequence>
                <xsd:element name="ident" type="cdf:identType" minOccurs="0"
                    maxOccurs="unbounded"/>
                <xsd:element name="impact-metric" type="xsd:string"
                    minOccurs="0" maxOccurs="1"/>
                <xsd:element name="profile-note" minOccurs="0"
                    type="cdf:profileNoteType" maxOccurs="unbounded"/>
                <xsd:element name="fixtext" type="cdf:fixTextType"
                    minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element name="fix" type="cdf:fixType" minOccurs="0"
                    maxOccurs="unbounded"/>
                <xsd:choice>
                    <xsd:element name="check" type="cdf:checkType"
                        minOccurs="0" maxOccurs="unbounded"/>
                    <xsd:element name="complex-check" minOccurs="0"
                        type="cdf:complexCheckType" maxOccurs="1"/>
                </xsd:choice>
                <xsd:element name="signature" type="cdf:signatureType"
                    minOccurs="0" maxOccurs="1"/>
            </xsd:sequence>
            <xsd:attribute name="role" type="cdf:roleEnumType"
```

```
                use="optional" default="full"/>
            <xsd:attribute name="severity" type="cdf:severityEnumType"
                default="unknown" use="optional"/>
            <xsd:attribute name="multiple" type="xsd:boolean" use="optional"
                default="false"/>
            <xsd:attribute name="multi-check" type="xsd:boolean"
                use="optional" default="false"/>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>


<!-- ************************************************************ -->
<!-- ****************  Rule-related Types ********************** -->
<!-- ************************************************************ -->
<xsd:complexType name="identType">
    <xsd:annotation>
        <xsd:documentation xml:lang="en"> Type for a long-term globally
            meaningful identifier, consisting of a string (ID) and a URI of
            the naming scheme within which the name is meaningful.
        </xsd:documentation>
    </xsd:annotation>
    <xsd:simpleContent>
        <xsd:extension base="xsd:string">
            <xsd:attribute name="system" type="xsd:anyURI" use="required"/>
        </xsd:extension>
    </xsd:simpleContent>
</xsd:complexType>

<xsd:complexType name="warningType" mixed="true">
    <xsd:annotation>
        <xsd:documentation xml:lang="en"> Data type for the warning element
            under the Rule object, a rich text string with substitutions
            allowed, plus an attribute for the kind of warning.
        </xsd:documentation>
    </xsd:annotation>
    <xsd:complexContent>
        <xsd:extension base="cdf:htmlTextWithSubType">
            <xsd:attribute name="category"
                type="cdf:warningCategoryEnumType" use="optional"
                default="general"/>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:simpleType name="warningCategoryEnumType">
    <xsd:annotation>
        <xsd:documentation xml:lang="en"> Allowed warning category keywords
            for the warning element. The allowed categories are:
            general=broad or general-purpose warning (default for
            compatibility for XCCDF 1.0) functionality=warning about
            possible impacts to functionality or operational features
            performance=warning about changes to target system performance
            or throughput hardware=warning about hardware restrictions or
            possible impacts to hardware legal=warning about legal
            implications regulatory=warning about regulatory obligations or
            compliance implications management=warning about impacts to the
            mgmt or administration of the target system audit=warning about
            impacts to audit or logging dependency=warning about
            dependencies between this Rule and other parts of the target
            system, or version dependencies. </xsd:documentation>
    </xsd:annotation>
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="general"/>
        <xsd:enumeration value="functionality"/>
```

```
            <xsd:enumeration value="performance"/>
            <xsd:enumeration value="hardware"/>
            <xsd:enumeration value="legal"/>
            <xsd:enumeration value="regulatory"/>
            <xsd:enumeration value="management"/>
            <xsd:enumeration value="audit"/>
            <xsd:enumeration value="dependency"/>
        </xsd:restriction>
    </xsd:simpleType>


    <xsd:complexType name="fixTextType" mixed="true">
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> Data type for the fixText element
                that represents a rich text string, with substitutions allowed,
                and a series of attributes that qualify the fix.
            </xsd:documentation>
        </xsd:annotation>
        <xsd:complexContent>
            <xsd:extension base="cdf:htmlTextWithSubType">
                <xsd:attribute name="fixref" type="xsd:NCName" use="optional"/>
                <xsd:attribute name="reboot" type="xsd:boolean" use="optional"
                    default="0"/>
                <xsd:attribute name="strategy" type="cdf:fixStrategyEnumType"
                    use="optional" default="unknown"/>
                <xsd:attribute name="disruption" type="cdf:ratingEnumType"
                    use="optional" default="unknown"/>
                <xsd:attribute name="complexity" type="cdf:ratingEnumType"
                    use="optional" default="unknown"/>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>

    <xsd:complexType name="fixType" mixed="true">
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> Type for a string with embedded
                Value and instance substitutions and an optional platform id ref
                attribute, but no embedded XHTML markup. The platform attribute
                should refer to a platform-definition element in the
                platform-definitions child of the Benchmark.
            </xsd:documentation>
        </xsd:annotation>
        <xsd:choice minOccurs="0" maxOccurs="unbounded">
            <xsd:element name="sub" type="cdf:idrefType"/>
            <xsd:element name="instance" type="cdf:instanceFixType"/>
        </xsd:choice>
        <xsd:attribute name="id" type="xsd:NCName" use="optional"/>
        <xsd:attribute name="reboot" type="xsd:boolean" use="optional"
            default="0"/>
        <xsd:attribute name="strategy" type="cdf:fixStrategyEnumType"
            use="optional" default="unknown"/>
        <xsd:attribute name="disruption" type="cdf:ratingEnumType"
            use="optional" default="unknown"/>
        <xsd:attribute name="complexity" type="cdf:ratingEnumType"
            use="optional" default="unknown"/>
        <xsd:attribute name="system" type="xsd:anyURI" use="optional"/>
        <xsd:attribute name="platform" type="xsd:anyURI" use="optional"/>
    </xsd:complexType>

    <xsd:simpleType name="fixStrategyEnumType">
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> Allowed strategy keyword values
                for a Rule fix or fixtext. The allowed values are: unknown=
                strategy not defined (default for forward compatibility for
                XCCDF 1.0) configure=adjust target config or settings
```

```
                patch=apply a patch, hotfix, or update policy=remediation by
                changing policies/procedures disable=turn off or deinstall
                something enable=turn on or install something restrict=adjust
                permissions or ACLs update=install upgrade or update the system
                combination=combo of two or more of the above
            </xsd:documentation>
        </xsd:annotation>
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="unknown"/>
            <xsd:enumeration value="configure"/>
            <xsd:enumeration value="combination"/>
            <xsd:enumeration value="disable"/>
            <xsd:enumeration value="enable"/>
            <xsd:enumeration value="patch"/>
            <xsd:enumeration value="policy"/>
            <xsd:enumeration value="restrict"/>
            <xsd:enumeration value="update"/>
        </xsd:restriction>
    </xsd:simpleType>

    <xsd:simpleType name="ratingEnumType">
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> Allowed rating values values for a
                Rule fix or fixtext: disruption, complexity, and maybe overhead.
                The possible values are: unknown= rating unknown or impossible
                to estimate (default for forward compatibility for XCCDF 1.0)
                low = little or no potential for disruption, very modest
                complexity medium= some chance of minor disruption, substantial
                complexity high = likely to cause serious disruption, very
                complex </xsd:documentation>
        </xsd:annotation>
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="unknown"/>
            <xsd:enumeration value="low"/>
            <xsd:enumeration value="medium"/>
            <xsd:enumeration value="high"/>
        </xsd:restriction>
    </xsd:simpleType>

    <xsd:complexType name="instanceFixType">
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> Type for an instance element in a
                fix element. The instance element inside a fix element
                designates a spot where the name of the instance should be
                substituted into the fix template to generate the final fix
                data. The instance element in this usage has one optional
                attribute: context. </xsd:documentation>
        </xsd:annotation>
        <xsd:attribute name="context" type="xsd:string" default="undefined"
            use="optional"/>
    </xsd:complexType>

    <xsd:complexType name="complexCheckType">
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> The type for an element that can
                contains a boolean expression based on checks. This element can
                have only complex-check and check elements as children. It has
                two attributes: operator and negate. The operator attribute can
                have values "OR" or "AND", and the negate attribute is boolean.
                See the specification document for truth tables for the
                operators and negations. Note: complex-check is defined in this
                way for conceptual equivalence with OVAL. </xsd:documentation>
        </xsd:annotation>
        <xsd:choice minOccurs="1" maxOccurs="unbounded">
            <xsd:element name="check" type="cdf:checkType"/>
```

```
            <xsd:element name="complex-check" type="cdf:complexCheckType"/>
        </xsd:choice>
        <xsd:attribute name="operator" type="cdf:ccOperatorEnumType"
            use="required"/>
        <xsd:attribute name="negate" default="0" type="xsd:boolean"
            use="optional"/>
    </xsd:complexType>

    <xsd:simpleType name="ccOperatorEnumType">
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> The type for the allowed operator
                names for the complex-check operator attribute. For now, we just
                allow boolean AND and OR as operators. (The complex-check has a
                separate mechanism for negation.) </xsd:documentation>
        </xsd:annotation>
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="OR"/>
            <xsd:enumeration value="AND"/>
        </xsd:restriction>
    </xsd:simpleType>

    <xsd:complexType name="checkType">
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> Data type for the check element, a
                checking system specification URI, and XML content. The content
                of the check element is: zero or more check-export elements,
                zero or more check-content-ref elements, and finally an optional
                check-content element. An content-less check element isn't
                legal, but XSD cannot express that! </xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:element name="check-import" type="cdf:checkImportType"
                minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element name="check-export" type="cdf:checkExportType"
                minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element name="check-content-ref" minOccurs="0"
                maxOccurs="unbounded" type="cdf:checkContentRefType"/>
            <xsd:element name="check-content" minOccurs="0" maxOccurs="1"
                type="cdf:checkContentType"/>
        </xsd:sequence>
        <xsd:attribute name="system" type="xsd:anyURI" use="required"/>
        <xsd:attribute name="negate" type="xsd:boolean" use="optional"
            default="false"/>
        <xsd:attribute name="id" type="xsd:NCName" use="optional"/>
        <xsd:attribute name="selector" default="" type="xsd:string"
            use="optional"/>
        <xsd:attribute ref="xml:base"/>
    </xsd:complexType>

    <xsd:complexType name="checkImportType" mixed="true">
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> Data type for the check-import
                element, which specifies a value that the benchmark author
                wishes to retrieve from the the checking system. The import-name
                attribute gives the name or id of the value in the checking
                system. When the check-import element appears in the context of
                a rule-result, then the element's content is the desired value.
                When the check-import element appears in the context of a Rule,
                then it should be empty and any content must be ignored. The
                import-xpath attribute allows further refinement if the imported
                value takes the form of XML structures. </xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:any processContents="skip" minOccurs="0"/>
        </xsd:sequence>
```

```
        <xsd:attribute name="import-name" type="xsd:string" use="required"/>
        <xsd:attribute name="import-xpath" type="xsd:string" use="optional"/>
    </xsd:complexType>

    <xsd:complexType name="checkExportType">
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> Data type for the check-export
                element, which specifies a mapping between an XCCDF internal
                Value id and a value name to be used by the checking system or
                processor. </xsd:documentation>
        </xsd:annotation>
        <xsd:attribute name="value-id" type="xsd:NCName" use="required"/>
        <xsd:attribute name="export-name" type="xsd:string" use="required"/>
    </xsd:complexType>

    <xsd:complexType name="checkContentRefType">
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> Data type for the
                check-content-ref element, which points to the code for a
                detached check in another file. This element has no body, just a
                couple of attributes: href and name. The name is optional, if it
                does not appear then this reference is to the entire other
                document. </xsd:documentation>
        </xsd:annotation>
        <xsd:attribute name="href" type="xsd:anyURI" use="required"/>
        <xsd:attribute name="name" type="xsd:string"/>
    </xsd:complexType>

    <xsd:complexType name="checkContentType" mixed="true">
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> Data type for the check-content
                element, which holds the actual code of an enveloped check in
                some other (non-XCCDF) language. This element can hold almost
                anything; XCCDF tools do not process its content directly.
            </xsd:documentation>
        </xsd:annotation>
        <xsd:choice minOccurs="0" maxOccurs="unbounded">
            <xsd:any namespace="##other" processContents="skip"/>
        </xsd:choice>
    </xsd:complexType>

    <xsd:simpleType name="weightType">
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> Data type for a Rule's weight, a
                non-negative real number. </xsd:documentation>
        </xsd:annotation>
        <xsd:restriction base="xsd:decimal">
            <xsd:minInclusive value="0.0"/>
            <xsd:totalDigits value="3"/>
        </xsd:restriction>
    </xsd:simpleType>

    <!-- ************************************************************* -->
    <!-- ******************  Value Element  ************************** -->
    <!-- ************************************************************* -->
    <xsd:element name="Value" type="cdf:valueType">
        <xsd:unique name="valueSelectorKey">
            <xsd:selector xpath="./cdf:value|cdf:complex-value"/>
            <xsd:field xpath="@selector"/>
        </xsd:unique>
        <xsd:unique name="defaultSelectorKey">
            <xsd:selector xpath="./cdf:default|cdf:complex-default"/>
            <xsd:field xpath="@selector"/>
        </xsd:unique>
        <xsd:unique name="matchSelectorKey">
```

```
            <xsd:selector xpath="./cdf:match"/>
            <xsd:field xpath="@selector"/>
        </xsd:unique>
        <xsd:unique name="lower-boundSelectorKey">
            <xsd:selector xpath="./cdf:lower-bound"/>
            <xsd:field xpath="@selector"/>
        </xsd:unique>
        <xsd:unique name="upper-boundSelectorKey">
            <xsd:selector xpath="./cdf:upper-bound"/>
            <xsd:field xpath="@selector"/>
        </xsd:unique>
        <xsd:unique name="choicesSelectorKey">
            <xsd:selector xpath="./cdf:choices"/>
            <xsd:field xpath="@selector"/>
        </xsd:unique>
    </xsd:element>

    <xsd:complexType name="valueType">
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> Data type for the Value element,
                which represents a tailorable string, boolean, or number in the
                Benchmark. </xsd:documentation>
        </xsd:annotation>
        <xsd:complexContent>
            <xsd:extension base="cdf:itemType">
                <xsd:sequence>
                    <xsd:choice minOccurs="1" maxOccurs="unbounded">
                        <xsd:element name="value" type="cdf:selStringType"
                            minOccurs="1" maxOccurs="1"/>
                        <xsd:element name="complex-value"
                            type="cdf:selComplexValueType" minOccurs="1"
                            maxOccurs="1"/>
                    </xsd:choice>
                    <xsd:choice minOccurs="0" maxOccurs="unbounded">
                        <xsd:element name="default" type="cdf:selStringType"
                            minOccurs="1" maxOccurs="1"/>
                        <xsd:element name="complex-default"
                            type="cdf:selComplexValueType" minOccurs="1"
                            maxOccurs="1"/>
                    </xsd:choice>
                    <xsd:element name="match" type="cdf:selStringType"
                        minOccurs="0" maxOccurs="unbounded"/>
                    <xsd:element name="lower-bound" type="cdf:selNumType"
                        minOccurs="0" maxOccurs="unbounded"/>
                    <xsd:element name="upper-bound" type="cdf:selNumType"
                        minOccurs="0" maxOccurs="unbounded"/>
                    <xsd:element name="choices" type="cdf:selChoicesType"
                        minOccurs="0" maxOccurs="unbounded"/>
                    <xsd:element name="source" type="cdf:uriRefType"
                        minOccurs="0" maxOccurs="unbounded"/>
                    <xsd:element name="signature" type="cdf:signatureType"
                        minOccurs="0" maxOccurs="1"/>
                </xsd:sequence>
                <xsd:attribute name="type" type="cdf:valueTypeType"
                    default="string" use="optional"/>
                <xsd:attribute name="operator" type="cdf:valueOperatorType"
                    default="equals" use="optional"/>
                <xsd:attribute name="interactive" type="xsd:boolean" default="0"
                    use="optional"/>
                <xsd:attribute name="interfaceHint" use="optional"
                    type="cdf:interfaceHintType"/>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
```

```
<!-- ************************************************************ -->
<!-- **************  Value-related Types  ********************** -->
<!-- ************************************************************ -->
<xsd:complexType name="complexValueType">
    <xsd:annotation>
        <xsd:documentation> The type that a Value may encapsulate. This can
            be a list or some external XML representation of a data
            structure. </xsd:documentation>
    </xsd:annotation>
    <xsd:choice>
        <xsd:element name="item" minOccurs="1" maxOccurs="unbounded"
            type="xsd:string"/>
        <!-- For lists -->
        <xsd:element name="external-type">
            <!-- For external structures -->
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:any minOccurs="1" maxOccurs="unbounded"
                        processContents="strict" namespace="##other"/>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
    </xsd:choice>
</xsd:complexType>

<xsd:complexType name="selComplexValueType">
    <xsd:annotation>
        <xsd:documentation> Use a selector to identify a complex value for
            later reference. </xsd:documentation>
    </xsd:annotation>
    <xsd:complexContent>
        <xsd:extension base="cdf:complexValueType">
            <xsd:attribute name="selector" default="" type="xsd:string"
                use="optional"/>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="selChoicesType">
    <xsd:annotation>
        <xsd:documentation xml:lang="en"> The choice element specifies a
            list of legal or suggested choices for a Value object. It holds
            one or more choice or complex-choice elements, a mustMatch
            attribute, and a selector attribute. </xsd:documentation>
    </xsd:annotation>
    <xsd:choice minOccurs="1" maxOccurs="unbounded">
        <xsd:element name="choice" type="xsd:string" minOccurs="1"
            maxOccurs="1"/>
        <xsd:element name="complex-choice" type="cdf:complexValueType"
            minOccurs="1" maxOccurs="1"/>
    </xsd:choice>
    <xsd:attribute name="mustMatch" type="xsd:boolean" use="optional"/>
    <xsd:attribute name="selector" default="" type="xsd:string"
        use="optional"/>
</xsd:complexType>

<xsd:complexType name="selStringType">
    <xsd:annotation>
        <xsd:documentation xml:lang="en"> This type is for an element that
            has string content and a selector attribute. It is used for some
            of the child elements of Value. </xsd:documentation>
    </xsd:annotation>
    <xsd:simpleContent>
        <xsd:extension base="xsd:string">
```

```
            <xsd:attribute name="selector" default="" type="xsd:string"
                use="optional"/>
        </xsd:extension>
    </xsd:simpleContent>
</xsd:complexType>

<xsd:complexType name="selNumType">
    <xsd:annotation>
        <xsd:documentation xml:lang="en"> This type is for an element that
            has numeric content and a selector attribute. It is used for two
            of the child elements of Value. </xsd:documentation>
    </xsd:annotation>
    <xsd:simpleContent>
        <xsd:extension base="xsd:decimal">
            <xsd:attribute name="selector" default="" type="xsd:string"
                use="optional"/>
        </xsd:extension>
    </xsd:simpleContent>
</xsd:complexType>

<xsd:complexType name="uriRefType">
    <xsd:annotation>
        <xsd:documentation xml:lang="en"> Data type for elements that have
            no content, just a URI. </xsd:documentation>
    </xsd:annotation>
    <xsd:attribute name="uri" type="xsd:anyURI" use="required"/>
</xsd:complexType>

<xsd:simpleType name="valueTypeType">
    <xsd:annotation>
        <xsd:documentation xml:lang="en"> Allowed data types for Values,
            just string, numeric, and true/false. </xsd:documentation>
    </xsd:annotation>
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="number"/>
        <xsd:enumeration value="string"/>
        <xsd:enumeration value="boolean"/>
    </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="valueOperatorType">
    <xsd:annotation>
        <xsd:documentation xml:lang="en"> Allowed operators for Values. Note
            that most of these are valid only for numeric data, but the
            schema doesn't enforce that. </xsd:documentation>
    </xsd:annotation>
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="equals"/>
        <xsd:enumeration value="not equal"/>
        <xsd:enumeration value="greater than"/>
        <xsd:enumeration value="less than"/>
        <xsd:enumeration value="greater than or equal"/>
        <xsd:enumeration value="less than or equal"/>
        <xsd:enumeration value="pattern match"/>
    </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="interfaceHintType">
    <xsd:annotation>
        <xsd:documentation xml:lang="en"> Allowed interface hint values.
            When an interfaceHint appears on the Value, it provides a
            suggestion to a tailoring or benchmarking tool about how to
            present the UI for adjusting a Value. </xsd:documentation>
    </xsd:annotation>
    <xsd:restriction base="xsd:string">
```

```
            <xsd:enumeration value="choice"/>
            <xsd:enumeration value="textline"/>
            <xsd:enumeration value="text"/>
            <xsd:enumeration value="date"/>
            <xsd:enumeration value="datetime"/>
        </xsd:restriction>
    </xsd:simpleType>

    <!-- ************************************************************ -->
    <!-- ******************  Profile Element  ********************** -->
    <!-- ************************************************************ -->
    <xsd:element name="Profile" type="cdf:profileType">
        <!-- selector key constraints -->
        <xsd:unique name="itemSelectKey">
            <xsd:selector xpath="./cdf:select"/>
            <xsd:field xpath="@idref"/>
        </xsd:unique>
        <xsd:unique name="refineRuleKey">
            <xsd:selector xpath="./cdf:refine-rule"/>
            <xsd:field xpath="@idref"/>
        </xsd:unique>
        <xsd:unique name="refineValueKey">
            <xsd:selector xpath="./cdf:refine-value"/>
            <xsd:field xpath="@idref"/>
        </xsd:unique>
        <xsd:unique name="setValueKey">
            <xsd:selector xpath="./cdf:set-value"/>
            <xsd:field xpath="@idref"/>
        </xsd:unique>
    </xsd:element>

    <xsd:complexType name="profileType">
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> Data type for the Profile element,
                which holds a specific tailoring of the Benchmark. The main part
                of a Profile is the selectors: select, set-value, refine-rule,
                and refine-value. A Profile may also be signed with an
                XML-Signature. </xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:element ref="cdf:status" minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element name="dc-status" minOccurs="0" maxOccurs="1"
                type="cdf:dc-statusType"/>
            <xsd:element name="version" type="cdf:versionType" minOccurs="0"
                maxOccurs="1"/>
            <xsd:element name="title" type="cdf:textWithSubType" minOccurs="1"
                maxOccurs="unbounded"/>
            <xsd:element name="description" type="cdf:htmlTextWithSubType"
                minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element name="reference" type="cdf:referenceType" minOccurs="0"
                maxOccurs="unbounded"/>
            <xsd:element name="platform" type="cdf:overrideableCPE2idrefType"
                minOccurs="0" maxOccurs="unbounded"/>
            <xsd:choice minOccurs="0" maxOccurs="unbounded">
                <xsd:element name="select" minOccurs="0"
                    type="cdf:profileSelectType"/>
                <xsd:element name="set-complex-value" minOccurs="0"
                    type="cdf:profileSetComplexValueType"/>
                <xsd:element name="set-value" minOccurs="0"
                    type="cdf:profileSetValueType"/>
                <xsd:element name="refine-value" minOccurs="0"
                    type="cdf:profileRefineValueType"/>
                <xsd:element name="refine-rule" minOccurs="0"
                    type="cdf:profileRefineRuleType"/>
            </xsd:choice>
```

```
            <xsd:element name="metadata" type="cdf:metadataType" minOccurs="0"
                maxOccurs="unbounded"/>
            <xsd:element name="signature" type="cdf:signatureType" minOccurs="0"
                maxOccurs="1"/>
        </xsd:sequence>
        <xsd:attribute name="id" type="xsd:NCName" use="required"/>
        <xsd:attribute name="prohibitChanges" type="xsd:boolean" default="false"
            use="optional"/>
        <xsd:attribute name="abstract" type="xsd:boolean" default="false"
            use="optional"/>
        <xsd:attribute name="note-tag" type="xsd:NCName" use="optional"/>
        <xsd:attribute name="extends" type="xsd:NCName" use="optional"/>
        <xsd:attribute ref="xml:base"/>
        <xsd:attribute name="Id" type="xsd:ID" use="optional"/>
    </xsd:complexType>

    <!-- ************************************************************ -->
    <!-- **************  Profile-related Types ********************** -->
    <!-- ************************************************************ -->
    <xsd:complexType name="profileSelectType">
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> Type for the select element in a
                Profile; all it has are two attributes, no content. The two
                attributes are idref which refers to a Group or Rule, and
                selected which is boolean. As content, the select element can
                contain zero or more remark elements, which allows the benchmark
                author to add explanatory material or other additional prose.
            </xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:element name="remark" type="cdf:textType" minOccurs="0"
                maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:attribute name="idref" type="xsd:NCName" use="required"/>
        <xsd:attribute name="selected" type="xsd:boolean" use="required"/>
    </xsd:complexType>

    <xsd:complexType name="profileSetValueType">
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> Type for the set-value element in
                a Profile; it has one required attribute and string content. The
                attribute is 'idref', it refers to a Value. </xsd:documentation>
        </xsd:annotation>
        <xsd:simpleContent>
            <xsd:extension base="xsd:string">
                <xsd:attribute name="idref" type="xsd:NCName" use="required"/>
            </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>

    <xsd:complexType name="profileSetComplexValueType">
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> Type for the set-complex-value
                element in a Profile; it has one required attribute and
                complexValueType content. The attribute is 'idref', it refers to
                a Value. </xsd:documentation>
        </xsd:annotation>
        <xsd:complexContent>
            <xsd:extension base="cdf:complexValueType">
                <xsd:attribute name="idref" type="xsd:NCName" use="required"/>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>

    <xsd:complexType name="profileRefineValueType">
```

```
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> Type for the refine-value element
                in a Profile; all it has are three attributes, no content. The
                three attributes are 'idref' which refers to a Value, 'selector'
                which designates certain element children of the Value, and
                'operator' which can override the operator attribute of the
                Value. As content, the refine-value element can contain zero or
                more remark elements, which allows the benchmark author to add
                explanatory material or other additional prose.
            </xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:element name="remark" type="cdf:textType" minOccurs="0"
                maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:attribute name="idref" type="xsd:NCName" use="required"/>
        <xsd:attribute name="selector" type="xsd:string" use="optional"/>
        <xsd:attribute name="operator" type="cdf:valueOperatorType"
            use="optional"/>
    </xsd:complexType>

    <xsd:complexType name="profileRefineRuleType">
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> Type for the refine-rule element
                in a Profile; all it has are four attributes, no content. The
                main attribute is 'idref' which refers to a Rule, and three
                attributes that allow the Profile author to adjust aspects of
                how a Rule is processed during a benchmark run: weight,
                severity, role. As content, the refine-rule element can contain
                zero or more remark elements, which allows the benchmark author
                to add explanatory material or other additional prose.
            </xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:element name="remark" type="cdf:textType" minOccurs="0"
                maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:attribute name="idref" type="xsd:NCName" use="required"/>
        <xsd:attribute name="weight" type="cdf:weightType" use="optional"/>
        <xsd:attribute name="selector" type="xsd:string" use="optional"/>
        <xsd:attribute name="severity" type="cdf:severityEnumType"
            use="optional"/>
        <xsd:attribute name="role" type="cdf:roleEnumType" use="optional"/>
    </xsd:complexType>

    <!-- ************************************************************ -->
    <!-- ******************  TestResult Element  ******************** -->
    <!-- ************************************************************ -->
    <xsd:element name="TestResult" type="cdf:testResultType"/>

    <xsd:complexType name="testResultType">
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> Data type for the TestResult
                element, which holds the results of one application of the
                Benchmark. The optional test-system attribute gives the name of
                the benchmarking tool. </xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:element name="benchmark" minOccurs="0" maxOccurs="1">
                <xsd:complexType>
                    <xsd:attribute name="href" type="xsd:anyURI" use="required"
                    />
                    <xsd:attribute name="id" type="xsd:NCName" use="optional"/>
                </xsd:complexType>
            </xsd:element>
```

```
            <xsd:element name="title" type="cdf:textType" minOccurs="0"
                maxOccurs="unbounded"/>
            <xsd:element name="remark" type="cdf:textType" minOccurs="0"
                maxOccurs="unbounded"/>
            <xsd:element name="organization" type="xsd:string" minOccurs="0"
                maxOccurs="unbounded"/>
            <xsd:element name="identity" type="cdf:identityType" minOccurs="0"
                maxOccurs="1"/>
            <xsd:element name="profile" type="cdf:idrefType" minOccurs="0"
                maxOccurs="1"/>
            <xsd:element name="target" type="xsd:string" minOccurs="1"
                maxOccurs="unbounded"/>
            <xsd:element name="target-address" type="xsd:string" minOccurs="0"
                maxOccurs="unbounded"/>
            <xsd:element name="target-facts" type="cdf:targetFactsType"
                minOccurs="0" maxOccurs="1"/>
            <xsd:element name="platform" type="cdf:CPE2idrefType" minOccurs="0"
                maxOccurs="unbounded"/>
            <xsd:choice minOccurs="0" maxOccurs="unbounded">
                <xsd:element name="set-value" type="cdf:profileSetValueType"
                    minOccurs="1" maxOccurs="1"/>
                <xsd:element name="set-complex-value"
                    type="cdf:profileSetComplexValueType" minOccurs="1"
                    maxOccurs="1"/>
            </xsd:choice>
            <xsd:element name="rule-result" type="cdf:ruleResultType"
                minOccurs="0" maxOccurs="unbounded">
                <!-- Each context name in an instance must be unique. -->
                <xsd:key name="instanceContextKey">
                    <xsd:selector xpath="cdf:instance"/>
                    <xsd:field xpath="@context"/>
                </xsd:key>
                <!-- parentContext must refer to valid sibling context -->
                <xsd:keyref name="parentKeyRef" refer="cdf:instanceContextKey">
                    <xsd:selector xpath="./cdf:instance"/>
                    <xsd:field xpath="@parentContext"/>
                </xsd:keyref>
            </xsd:element>
            <xsd:element name="score" type="cdf:scoreType" minOccurs="1"
                maxOccurs="unbounded"/>
            <xsd:element name="metadata" type="cdf:metadataType" minOccurs="0"
                maxOccurs="unbounded"/>
            <xsd:element name="signature" type="cdf:signatureType" minOccurs="0"
                maxOccurs="1"/>
        </xsd:sequence>
        <xsd:attribute name="id" type="xsd:NCName" use="required"/>
        <xsd:attribute name="start-time" type="xsd:dateTime" use="optional"/>
        <xsd:attribute name="end-time" type="xsd:dateTime" use="required"/>
        <xsd:attribute name="test-system" type="xsd:string" use="optional"/>
        <xsd:attribute name="version" type="xsd:string" use="optional"/>
        <xsd:attribute name="Id" type="xsd:ID" use="optional"/>
    </xsd:complexType>

    <xsd:complexType name="scoreType">
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> Type for a score value in a
                TestResult, the content is a real number and the element can
                have two optional attributes. </xsd:documentation>
        </xsd:annotation>
        <xsd:simpleContent>
            <xsd:extension base="xsd:decimal">
                <xsd:attribute name="system" type="xsd:anyURI" use="optional"/>
                <xsd:attribute name="maximum" type="xsd:decimal" use="optional"
                />
            </xsd:extension>
```

```
        </xsd:simpleContent>
    </xsd:complexType>

    <xsd:complexType name="targetFactsType">
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> This element holds a list of facts
                about the target system or platform. Each fact is an element of
                type factType. Each fact must have a name, but duplicate names
                are allowed. (For example, if you had a fact about MAC
                addresses, and the target system had three NICs, then you'd need
                three instance of the "urn:xccdf:fact:ethernet:MAC" fact.)
            </xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:element name="fact" type="cdf:factType" minOccurs="0"
                maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="identityType">
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> Type for an identity element in a
                TestResult. The content is a string, the name of the identity.
                The authenticated attribute indicates whether the test system
                authenticated using that identity in order to apply the
                benchmark. The privileged attribute indicates whether the
                identity has access rights beyond those of normal system users
                (e.g. "root" on Unix") </xsd:documentation>
        </xsd:annotation>
        <xsd:simpleContent>
            <xsd:extension base="xsd:string">
                <xsd:attribute name="authenticated" type="xsd:boolean"
                    use="required"/>
                <xsd:attribute name="privileged" type="xsd:boolean"
                    use="required"/>
            </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>

    <xsd:complexType name="factType">
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> Element type for a fact about a
                target system: a name-value pair with a type. The content of the
                element is the value, the type attribute gives the type. This is
                an area where XML schema is weak: we can't make the schema
                validator check that the content matches the type.
            </xsd:documentation>
        </xsd:annotation>
        <xsd:simpleContent>
            <xsd:extension base="xsd:string">
                <xsd:attribute name="name" type="xsd:anyURI" use="required"/>
                <xsd:attribute name="type" type="cdf:valueTypeType"
                    default="boolean" use="optional"/>
            </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>

    <xsd:complexType name="ruleResultType">
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> This element holds all the
                information about the application of one rule to a target. It
                may only appear as part of a TestResult object.
            </xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
```

```
            <xsd:element name="result" type="cdf:resultEnumType" minOccurs="1"
                maxOccurs="1"/>
            <xsd:element name="override" type="cdf:overrideType" minOccurs="0"
                maxOccurs="unbounded"/>
            <xsd:element name="ident" type="cdf:identType" minOccurs="0"
                maxOccurs="unbounded"/>
            <xsd:element name="metadata" type="cdf:metadataType" minOccurs="0"
                maxOccurs="unbounded"/>
            <xsd:element name="message" type="cdf:messageType" minOccurs="0"
                maxOccurs="unbounded"/>
            <xsd:element name="instance" type="cdf:instanceResultType"
                minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element name="fix" type="cdf:fixType" minOccurs="0"
                maxOccurs="unbounded"/>
            <!-- will we need a new restricted form for this? -->
            <xsd:element name="check" type="cdf:checkType" minOccurs="0"
                maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:attribute name="idref" type="xsd:NCName" use="required"/>
        <xsd:attribute name="role" type="cdf:roleEnumType" use="optional"/>
        <xsd:attribute name="severity" type="cdf:severityEnumType"
            use="optional"/>
        <xsd:attribute name="time" type="xsd:dateTime" use="optional"/>
        <xsd:attribute name="version" type="xsd:string" use="optional"/>
        <xsd:attribute name="weight" type="cdf:weightType" use="optional"/>
    </xsd:complexType>

    <xsd:complexType name="instanceResultType">
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> Type for an instance element in a
                rule-result. The content is a string, but the element may also
                have two attribute: context and parentContext. This type records
                the details of the target system instance for multiply
                instantiated rules. </xsd:documentation>
        </xsd:annotation>
        <xsd:simpleContent>
            <xsd:extension base="xsd:string">
                <xsd:attribute name="context" default="undefined"
                    type="xsd:string" use="optional"/>
                <xsd:attribute name="parentContext" type="xsd:string"
                    use="optional"/>
            </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>

    <xsd:complexType name="overrideType">
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> Type for an override block in a
                rule-result. It contains five mandatory parts: time, authority,
                old-result, new-result, and remark. </xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:element name="old-result" type="cdf:resultEnumType"
                minOccurs="1" maxOccurs="1"/>
            <xsd:element name="new-result" type="cdf:resultEnumType"
                minOccurs="1" maxOccurs="1"/>
            <xsd:element name="remark" type="cdf:textType" minOccurs="1"
                maxOccurs="1"/>
        </xsd:sequence>
        <xsd:attribute name="time" type="xsd:dateTime" use="required"/>
        <xsd:attribute name="authority" type="xsd:string" use="required"/>
    </xsd:complexType>

    <xsd:complexType name="messageType">
        <xsd:annotation>
```

```
            <xsd:documentation xml:lang="en"> Type for a message generated by
                the checking engine or XCCDF tool during benchmark testing.
                Content is string plus required severity attribute.
            </xsd:documentation>
        </xsd:annotation>
        <xsd:simpleContent>
            <xsd:extension base="xsd:string">
                <xsd:attribute name="severity" type="cdf:msgSevEnumType"
                    use="required"/>
            </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>

    <xsd:simpleType name="msgSevEnumType">
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> Allowed values for message
                severity. </xsd:documentation>
        </xsd:annotation>
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="error"/>
            <xsd:enumeration value="warning"/>
            <xsd:enumeration value="info"/>
        </xsd:restriction>
    </xsd:simpleType>

    <xsd:simpleType name="resultEnumType">
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> Allowed result indicators for a
                test, several possibilities: pass= the test passed, target
                complies w/ benchmark fail= the test failed, target does not
                comply error= an error occurred and test could not complete, or
                the test does not apply to this plaform unknown= could not tell
                what happened, results with this status are not to be scored
                notapplicable=Rule did not apply to test target fixed=rule
                failed, but was later fixed (score as pass) notchecked=Rule did
                not cause any evaluation by the checking engine (role of
                "unchecked") notselected=Rule was not selected in the Benchmark,
                and therefore was not checked (selected="0") informational=Rule
                was evaluated by the checking engine, but isn't to be scored
                (role of "unscored") </xsd:documentation>
        </xsd:annotation>
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="pass"/>
            <xsd:enumeration value="fail"/>
            <xsd:enumeration value="error"/>
            <xsd:enumeration value="unknown"/>
            <xsd:enumeration value="notapplicable"/>
            <xsd:enumeration value="notchecked"/>
            <xsd:enumeration value="notselected"/>
            <xsd:enumeration value="informational"/>
            <xsd:enumeration value="fixed"/>
        </xsd:restriction>
    </xsd:simpleType>

    <xsd:simpleType name="severityEnumType">
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> Allowed severity values for a
                Rule. there are several possible values: unknown= severity not
                defined (default, for forward compatibility from XCCDF 1.0) info
                = rule is informational only, failing the rule does not imply
                failure to conform to the security guidance of the benchmark.
                (usually would also have a weight of 0) low = not a serious
                problem medium= fairly serious problem high = a grave or
                critical problem </xsd:documentation>
        </xsd:annotation>
```

```
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="unknown"/>
            <xsd:enumeration value="info"/>
            <xsd:enumeration value="low"/>
            <xsd:enumeration value="medium"/>
            <xsd:enumeration value="high"/>
        </xsd:restriction>
    </xsd:simpleType>

    <xsd:simpleType name="roleEnumType">
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> Allowed checking and scoring roles
                for a Rule. There are several possible values: full = if the
                rule is selected, then check it and let the result contribute to
                the score and appear in reports (default, for compatibility for
                XCCDF 1.0). unscored = check the rule, and include the results
                in any report, but do not include the result in score
                computations (in the default scoring model the same effect can
                be achieved with weight=0) unchecked = don't check the rule,
                just force the result status to 'unknown'. Include the rule's
                information in any reports. </xsd:documentation>
        </xsd:annotation>
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="full"/>
            <xsd:enumeration value="unscored"/>
            <xsd:enumeration value="unchecked"/>
        </xsd:restriction>
    </xsd:simpleType>

</xsd:schema>
```

Testing

The XCCDF 1.2 schema has been checked for syntax and tested with the Oxygen 11.2 schema-validating parser using Saxon EE 9.2.0.6.

Appendix B. Sample Benchmark File

The sample below illustrates some of the concepts of XCCDF. It gives a few simple rules about configuration of a Cisco IOS router, based on material from the publicly available NSA Router Security Configuration Guide.

```
<?xml version="1.0"  encoding="UTF-8"?>
<cdf:Benchmark id="ios-test-1" resolved="0" xml:lang="en" style="sample"
   xmlns:cdf="http://checklists.nist.gov/xccdf/1.2"
   xmlns:cpe="http://cpe.mitre.org/language/2.0"
   xmlns:dc="http://purl.org/dc/elements/1.1/"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:htm="http://www.w3.org/1999/xhtml"
   xmlns:dsig="http://www.w3.org/2000/09/xmldsig#"
   xsi:schemaLocation="http://checklists.nist.gov/xccdf/1.2 xccdf-1.2.xsd
http://cpe.mitre.org/language/2.0 cpe-language_2.3.xsd">

   <cdf:status date="2010-06-01">draft</cdf:status>
   <cdf:title>XCCDF Sample for Cisco IOS</cdf:title>
   <cdf:description>
     This document defines a small set of rules for securing Cisco
     IOS routers. The set of rules constitute a <htm:i>benchmark</htm:i>.
     A benchmark usually represents an industry consensus of best
     practices.  It lists steps to be taken as well as rationale for
     them.  This example benchmark is merely a small subset of the
     rules that would be necessary for securing an IOS router.
   </cdf:description>

   <cdf:notice id="Sample-Terms-Of-Use" xml:lang="en">
      This document may be copied and used subject to the
      subject to the NIST terms of use
      (http://www.nist.gov/public_affairs/disclaim.htm)
      and the NSA Legal Notices
      (http://www.nsa.gov/notices/notic00004.cfm?Address=/).
   </cdf:notice>
   <cdf:front-matter>
     <htm:p>
       This benchmark assumes that you are running IOS 11.3 or later.
     </htm:p>
   </cdf:front-matter>
   <cdf:reference href="http://www.nsa.gov/ia/">
     NSA Router Security Configuration Guide, Version 1.1c
   </cdf:reference>
   <cdf:reference>
      <dc:title>Hardening Cisco Routers</dc:title>
      <dc:creator>Thomas Akin</dc:creator>
      <dc:publisher>O'Reilly and Associates</dc:publisher>
      <dc:identifier>http://www.ora.com/</dc:identifier>
   </cdf:reference>

   <cdf:plain-text id="os-name">
     Cisco Internet Operating System (tm)
   </cdf:plain-text>

   <cpe:platform-specification>
```

```
    <cpe:platform id="">
      <cpe:title>Cisco IOS 12.3 on Catalyst 6500 platform</cpe:title>
      <cpe:logical-test operator="AND" negate="0">
        <cpe:fact-ref name="cpe:/o:cisco:ios:12.3"/>
        <cpe:fact-ref name="cpe:/h:cisco:catalyst:6500"/>
      </cpe:logical-test>
    </cpe:platform>
  </cpe:platform-specification>

  <cdf:platform idref="cpe:/o:cisco:ios:12.3"/>
  <cdf:platform idref="cpe:/o:cisco:ios:12.2"/>
  <cdf:platform idref="cpe:/o:cisco:ios:12.1"/>
  <cdf:platform idref="cpe:/o:cisco:ios:12.0"/>
  <cdf:platform idref="cpe:/o:cisco:ios:11.3"/>
  <cdf:version>0.1.15</cdf:version>
  <cdf:model system="urn:xccdf:scoring:default"/>
  <cdf:model system="urn:xccdf:scoring:flat"/>
  <cdf:model system="urn:testing.com:scoring:relative">
     <cdf:param name="floor">0.0</cdf:param>
     <cdf:param name="ceiling">1000</cdf:param>
  </cdf:model>
 <cdf:Profile id="profile1" prohibitChanges="1" note-tag="lenient">
    <cdf:title>Sample Profile No. 1</cdf:title>
    <cdf:select idref="mgmt-plane" selected="0"/>
    <cdf:select idref="ctrl-plane" selected="1"/>
    <cdf:select idref="finger" selected="1"/>
    <cdf:set-value idref="exec-timeout-time">30</cdf:set-value>
    <cdf:refine-value idref="buffered-logging-level"
                  selector="lenient"/>
 </cdf:Profile>
 <cdf:Profile id="profile2" extends="profile1" note-tag="strict">
    <cdf:title override="1">Sample Profile No. 2</cdf:title>
    <cdf:select idref="mgmt-plane" selected="1"/>
    <cdf:select idref="data-plane" selected="1"/>
    <cdf:set-value idref="exec-timeout-time">10</cdf:set-value>
    <cdf:refine-rule idref="no-tcp-small-servers"
                    weight="0.8" severity="medium"/>
    <cdf:refine-value idref="buffered-logging-level" selector="strict">
       <cdf:remark>Use strict logging for this profile</cdf:remark>
    </cdf:refine-value>
 </cdf:Profile>

 <cdf:Value id="exec-timeout-time" type="number"
            operator="less than or equal">
   <cdf:title>IOS - line exec timeout value</cdf:title>
   <cdf:description>
      The length of time, in minutes, that an interactive session
      should be allowed to stay idle before being terminated.
   </cdf:description>
   <cdf:question>Session exec timeout time (in minutes)</cdf:question>
   <cdf:value>10</cdf:value>
   <cdf:default>15</cdf:default>
   <cdf:lower-bound>1</cdf:lower-bound>
   <cdf:upper-bound>60</cdf:upper-bound>
 </cdf:Value>
```

```
  <cdf:Group id="mgmt-plane" selected="1" prohibitChanges="1" weight="3">
     <cdf:title>Management Plane Rules</cdf:title>
     <cdf:description>
       Services, settings, and data streams related tosetting up
       and examining the static configuration of the router, and the
       authentication and authorization of administrators/operators.
     </cdf:description>
     <cdf:requires idref="no-directed-broadcast no-tcp-small-servers"/>
     <cdf:Rule id="no-finger-service-base" selected="0" weight="5.0"
               prohibitChanges="1" hidden="1"
               abstract="1" cluster-id="finger">
       <cdf:title>IOS - no IP finger service</cdf:title>
       <cdf:description>
         Disable the finger service, it can reveal information
         about logged in users to unauthorized parties.
         (For <cdf:sub idref="os-name"/> version 11.3 and later.)
       </cdf:description>
       <cdf:question>Prohibit the finger service</cdf:question>
       <cdf:ident system="http:/cce.mitre.org/">CCE-12345</cdf:ident>
             <cdf:fixtext fixref="no-finger" xml:lang="en">
          Turn off the finger service altogether,
          it is <htm:i>very</htm:i> rarely used.
       </cdf:fixtext>
       <cdf:check system="http://oval.mitre.org/XMLSchema/oval">
         <cdf:check-content-ref
href="http://oval.mitre.org/repository/find?file=iosDefns.xml"
name="OVAL1002"/>
         <cdf:check-content-ref href="iosDefns.xml" name="OVAL1002"/>
       </cdf:check>
     </cdf:Rule>

     <cdf:Rule id="ios11-no-finger-service"
               selected="0" prohibitChanges="1"
               hidden="0" weight="5" extends="no-finger-service-base">
       <cdf:title override="1">IOS 11 - no IP finger service</cdf:title>
       <cdf:platform idref="cpe:/o:cisco:ios:11.3"/>
       <cdf:fix id="no-finger" system="urn:xccdf:fix:system:commands"
                disruption="low" strategy="disable">
           no service finger
       </cdf:fix>
     </cdf:Rule>

     <cdf:Rule id="ios12-no-finger-service"
               selected="0" prohibitChanges="1"
               hidden="0" weight="5" extends="no-finger-service-base">
       <cdf:title override="1">IOS 12 - no IP finger service</cdf:title>
       <cdf:platform idref="cpe:/o:cisco:ios:12.3"/>
       <cdf:platform idref="cpe:/o:cisco:ios:12.2"/>
       <cdf:platform idref="cpe:/o:cisco:ios:12.1"/>
       <cdf:platform idref="cpe:/o:cisco:ios:12.0"/>
       <cdf:fix id="no-finger" system="urn:xccdf:fix:system:commands"
                disruption="low" strategy="disable">
            no ip finger
       </cdf:fix>
     </cdf:Rule>
```

```
    <cdf:Rule id="req-exec-timeout" selected="1" weight="8" multiple="1">
      <cdf:title>Require exec timeout on admin sessions</cdf:title>
      <cdf:description>
        Configure each administrative access line to terminate idle
        sessions after a fixed period of time determined by local policy
      </cdf:description>
      <cdf:question>Require admin session idle timeout</cdf:question>
      <cdf:profile-note tag="lenient">
        Half an hour
      </cdf:profile-note>
      <cdf:profile-note tag="strict">
        Ten minutes or less
      </cdf:profile-note>
      <cdf:fix strategy="configure" disruption="low"
               system="urn:xccdf:fix:commands">
          line vty 0 4
          exec-timeout <cdf:sub idref="exec-timeout-time"/>
      </cdf:fix>
      <cdf:check system="http://oval.mitre.org/XMLSchema/oval" id="foo">
        <cdf:check-export value-id="exec-timeout-time"
                          export-name="var-2"/>
        <cdf:check-content-ref href="iosDefns.xml" name="OVAL708"/>
      </cdf:check>
    </cdf:Rule>
  </cdf:Group>

  <cdf:Group id="ctrl-plane" selected="1" prohibitChanges="1" weight="3">
    <cdf:title>Control Plane Rules</cdf:title>
    <cdf:description>
         Services, settings, and data streams that support the
         operation and dynamic status of the router.
    </cdf:description>
    <cdf:question>Check rules related to system control</cdf:question>

    <cdf:Value id="buffered-logging-level" type="string"
               operator="equals" prohibitChanges="0"
               interfaceHint="choice">
      <cdf:title>Logging level for buffered logging</cdf:title>
      <cdf:description>
         Logging level for buffered logging; this setting is
         a severity level.  Every audit message of this
         severity or more (worse) will be logged.
      </cdf:description>
      <cdf:question>Select a buffered logging level</cdf:question>
      <cdf:value selector="strict">informational</cdf:value>
      <cdf:value selector="lenient">warning</cdf:value>
      <cdf:value>notification</cdf:value>
      <cdf:choices mustMatch="1">
         <cdf:choice>warning</cdf:choice>
         <cdf:choice>notification</cdf:choice>
         <cdf:choice>informational</cdf:choice>
      </cdf:choices>
      <cdf:source uri="urn:OS:Cisco:IOS:logging:levels"/>
    </cdf:Value>

    <cdf:Rule id="no-tcp-small-servers" selected="1"
```

```
            prohibitChanges="1" weight="7">
    <cdf:title>Disable tcp-small-servers</cdf:title>
    <cdf:description>
        Disable unnecessary services such as echo, chargen, etc.
    </cdf:description>
    <cdf:question>Prohibit TCP small services</cdf:question>
    <cdf:fixtext>
        Disable TCP small servers in IOS global config mode.
    </cdf:fixtext>
    <cdf:fix>no service tcp-small-servers</cdf:fix>
    <cdf:check system="http://oval.mitre.org/XMLSchema/oval">
       <cdf:check-content-ref href="iosDefns.xml" name="OVAL1000"/>
    </cdf:check>
  </cdf:Rule>

  <cdf:Rule id="no-udp-small-servers" selected="1" role="full"
            prohibitChanges="1" weight="5.7">
    <cdf:title>Disable udp-small-servers</cdf:title>
    <cdf:description>
        Disable unnecessary UDP services such as echo, chargen, etc.
    </cdf:description>
    <cdf:question>Forbid UDP small services</cdf:question>
    <cdf:fixtext>
      Disable UDP small servers in IOS global config mode.
    </cdf:fixtext>
    <cdf:fix>no service udp-small-servers</cdf:fix>
    <cdf:check system="http://oval.mitre.org/XMLSchema/oval">
       <cdf:check-content-ref href="iosDefns.xml" name="OVAL1001"/>
    </cdf:check>
  </cdf:Rule>

  <cdf:Rule id="enabled-buffered-logging-at-level" selected="1"
            prohibitChanges="0" weight="8.5">
    <cdf:title xml:lang="en">
       Ensure buffered logging enabled at proper level
    </cdf:title>
    <cdf:description>
       Make sure that buffered logging is enabled, and that
       the buffered logging level to one of the appropriate
       levels, Warning or higher.
    </cdf:description>
    <cdf:question>Check buffered logging and level</cdf:question>
    <cdf:fix>
       logging on
       logging buffered <cdf:sub idref="buffered-logging-level"/>
    </cdf:fix>
    <cdf:complex-check operator="AND" negate="1">
          <cdf:check system="http://oval.mitre.org/XMLSchema/oval">
               <cdf:check-export value-id="buffered-logging-level"
                     export-name="var-4"/>
               <cdf:check-content-ref href="iosDefns.xml"
                   name="org.cisecurity.cisco.ios.logging.buf.level"/>
          </cdf:check>
          <cdf:check system="http://oval.mitre.org/XMLSchema/oval">
               <cdf:check-content-ref href="iosDefns.xml"
                   name="org.cisecurity.cisco.ios.logging.enabled"/>
```

```
            </cdf:check>
       </cdf:complex-check>
     </cdf:Rule>
  </cdf:Group>

  <cdf:Group id="data-plane" selected="1" prohibitChanges="1" weight="2">
     <cdf:title>Data Plane Level 1</cdf:title>
     <cdf:description>
          Services and settings related to the data passing through
          the router (as opposed to directed to it). Basically, the
          data plane is for everything not in control or mgmt planes.
     </cdf:description>
     <cdf:question>Check rules related to data flow</cdf:question>

    <cdf:Group id="routing-rules" selected="1" prohibitChanges="1">
      <cdf:title>Routing Rules</cdf:title>
      <cdf:description>
         Rules in this group affect traffic forwarded through the
         router, including router actions taken on receipt of
         special data traffic.
      </cdf:description>
      <cdf:question>Apply standard forwarding protections</cdf:question>

    <cdf:Rule id="no-directed-broadcast" weight="7" multiple="1"
              selected="1" prohibitChanges="1">
      <cdf:title>IOS - no directed broadcasts</cdf:title>
      <cdf:description>
          Disable IP directed broadcast on each interface.
      </cdf:description>
      <cdf:question>Forbid IP directed broadcast</cdf:question>
      <cdf:fixtext>
        Disable IP directed broadcast on each interface
        using IOS interface configuration mode.
      </cdf:fixtext>
      <cdf:fix>
         interface <cdf:instance context="interface"/>
           no ip directed-broadcast
      </cdf:fix>
      <cdf:check system="http://oval.mitre.org/XMLSchema/oval">
         <cdf:check-content-ref href="iosDefns.xml" name="OVAL1101"/>
      </cdf:check>
    </cdf:Rule>
   </cdf:Group>
 </cdf:Group>

  <cdf:TestResult id="ios-test-5"
          end-time="2004-09-25T13:45:02-04:00">
    <cdf:benchmark href="ios-sample-v1.1.xccdf.xml"/>
    <cdf:title>Sample Results Block</cdf:title>
    <cdf:remark>Test run by Bob on Sept 25</cdf:remark>
    <cdf:organization>U.S. Government</cdf:organization>
    <cdf:organization>Department of Commerce</cdf:organization>
    <cdf:organization>National Institute of Standards and Technology
    </cdf:organization>
    <cdf:identity authenticated="1"
privileged="1">admin_bob</cdf:identity>
```

```
    <cdf:target>lower.test.net</cdf:target>
    <cdf:target-address>192.168.248.1</cdf:target-address>
    <cdf:target-address>2001:8::1</cdf:target-address>
    <cdf:target-facts>
        <cdf:fact type="string" name="urn:scap:fact:asset:identifier:mac">
            02:50:e6:c0:14:39
        </cdf:fact>
        <cdf:fact type="string"
name="urn:scap:fact:asset:identifier:host_name">
            lower
        </cdf:fact>
        <cdf:fact type="string"
name="urn:scap:fact:asset:identifier:ipv4">
            192.168.248.1
        </cdf:fact>
        <cdf:fact type="string"
name="urn:scap:fact:asset:identifier:ipv6">
            2001:8::1
        </cdf:fact>
    </cdf:target-facts>
    <cdf:set-value idref="exec-timeout-time">10</cdf:set-value>
    <cdf:rule-result idref ="ios12-no-finger-service"
                     time="2004-09-25T13:45:00-04:00">
        <cdf:result>pass</cdf:result>
    </cdf:rule-result>
    <cdf:rule-result idref ="req-exec-timeout"
                     time="2004-09-25T13:45:06-04:00">
        <cdf:result>pass</cdf:result>
        <cdf:override time="2004-09-25T13:59:00-04:00"
                  authority="Neal Ziring">
          <cdf:old-result>fail</cdf:old-result>
          <cdf:new-result>pass</cdf:new-result>
          <cdf:remark>Test override</cdf:remark>
        </cdf:override>
        <cdf:instance context="line">console</cdf:instance>
        <cdf:fix>
             line console
             exec-timeout 10 0
        </cdf:fix>
    </cdf:rule-result>
    <cdf:rule-result idref="ios12-no-finger-service">
      <cdf:result>notselected</cdf:result>
    </cdf:rule-result>
    <cdf:score system="urn:xccdf:model:default">67.5</cdf:score>
    <cdf:score system="urn:xccdf:model:flat" maximum="214">
       157.5
    </cdf:score>
  </cdf:TestResult>

</cdf:Benchmark>
```

Appendix C. Pre-Defined URIs

The following URLs and URNs are defined for XCCDF 1.2.

Long-Term Identification Systems

These URIs may appear as the value of the system attribute of an ident element (see p. 70).

`http://cve.mitre.org/`

MITRE's Common Vulnerabilities and Exposures – the identifier value should be a CVE number or CVE candidate number.

`http://cce.mitre.org/`

This specifies the Common Configuration Enumeration identifier scheme.

`http://www.cert.org/`

CERT Coordination Center – the identifier value should be a CERT advisory identifier (e.g. "CA-2004-02").

`http://www.us-cert.gov/cas/techalerts/`

US-CERT technical cyber security alerts – the identifier value should be a technical cyber security alert ID (e.g. "TA05-189A")

`http://www.kb.cert.org/`

US-CERT vulnerability notes database – the identifier values should be a vulnerability note number (e.g. "709220").

`http://iase.disa.mil/IAalerts/`

DISA Information Assurance Vulnerability Alerts (IAVA) – the identifier value should be a DOD IAVA identifier.

Check Systems

These are some URIs may appear as the value of the system attribute of a check element (see p. 60).

`http://oval.mitre.org/XMLSchema/oval`

MITRE's Open Vulnerability Assessment Language (see [16]).

`http://www.mitre.org/ocil/2`

MITRE's Open Checklist Interactive Language (see [8]).

`http://www.cisecurity.org/xccdf/interactive/1.0`

Center for Internet Security interactive query check system, used for asking the user questions about the target system during application of a security guidance document.

Scoring Models

These URIs may appear as the value of the system attribute on the model element or a score element (see pages 74 and 83).

`urn:xccdf:scoring:default`

This specifies the default (XCCDF 1.0) scoring model.

urn:xccdf:scoring:flat

This specifies the flat, weighted scoring model.

urn:xccdf:scoring:flat-unweighted

This specifies the flat scoring model with non-zero weights ignored (all non-zero weights set to 1).

urn:xccdf:scoring:absolute

This specifies the absolute (1 or 0) scoring model.

Target Platform Facts

The following URNs should be used to record facts about an IT asset (target) to which a Benchmark TestResult applies (see pages 57 and 87).

urn:scap:fact:asset:identifier:mac

Ethernet media access control address (should be sent as a pair with the IP or IPv6 address to ensure uniqueness)

urn:scap:fact:asset:identifier:ipv4

Internet Protocol version 4 address

urn:scap:fact:asset:identifier:ipv6

Internet Protocol version 6 address

urn:scap:fact:asset:identifier:host_name

Host name of the asset, if assigned

urn:scap:fact:asset:identifier:fqdn

Fully qualified domain name

urn:scap:fact:asset:identifier:ein

Equipment identification number or other inventory tag number

urn:scap:fact:asset:identifier:pki:

X.509 PKI certificate for the asset (encoded in Base-64)

urn:scap:fact:asset:identifier:pki:thumbprint

SHA.1 hash of the PKI certification for the asset (encoded in Base-64)

urn:scap:fact:asset:identifier:guid

Globally unique identifier for the asset

urn:scap:fact:asset:identifier:ldap

LDAP directory string (distinguished name) of the asset, if assigned

urn:scap:fact:asset:identifier:active_directory

Active Directory realm to which the asset belongs, if assigned

urn:scap:fact:asset:identifier:nis_domain

NIS domain of the asset, if assigned

urn:scap:fact:asset:environmental_information:owning_organization

Organization that tracks the asset on its inventory

urn:scap:fact:asset:environmental_information:current_region

Geographic region where the asset is located

urn:scap:fact:asset:environmental_information:administration_unit

Name of the organization that does system administration for the asset

urn:scap:fact:asset:environmental_information:administration_poc:title

Title (e.g., Mr, Ms, Col) of the system administrator for an asset]

urn:scap:fact:asset:environmental_information:administration_poc:e-mail

E-mail address of the system administrator for the asset

urn:scap:fact:asset:environmental_information:administration_poc:first_name

First name of the system administrator for the asset

urn:scap:fact:asset:environmental_information:administration_poc:last_name

Last name of the system administrator for the asset

Remediation Systems

The URIs represent remediation sources, mechanisms, schemes, or providers. They may appear as the system attribute on a fix element (see p. 68).

urn:xccdf:fix:commands

This specifies that the content of the fix element is a list of target system commands; executed in order, the commands should bring the target system into compliance with the Rule.

urn:xccdf:fix:urls

This specifies that the content of the fix element is a list of one or more URLs. The resources identified by the URLs should be applied to bring the system into compliance with the Rule.

urn:xccdf:fix:script:*language*

A URN of this form specifies that the content of the fix element is a script written in the given *language*. Executing the script should bring the target system into compliance with the Rule. The following languages are pre-defined:

- **sh** – Bourne shell
- **csh** – C Shell
- **perl** – Perl
- **batch** – Windows batch script
- **python** – Python and all Python-based scripting languages
- **vbscript** – Visual Basic Script (VBS)
- **javascript** – Javascript (ECMAScript, JScript)

- **`tcl`** – Tcl and all Tcl-based scripting languages

`urn:xccdf:fix:patch:`*`vendor`*

A URN of this form specifies that the content of the fix element is a patch identifier, in proprietary format as defined by the vendor. The vendor string should be the DNS domain name of the vendor, as defined in the CPE specification [17]. For example, for Microsoft Corporation, the DNS domain is "microsoft.com", and the CPE vendor name would be "microsoft".

Appendix D. References

[1] Fallside, David C., *XML Schema Part 0: Primer*, W3C Recommendation, May 2001. (http://www.w3.org/)

[2] Biron, Paul V. and Malhotra, Ashok, *XML Schema Part 2: Datatypes*, W3C Recommendation, May 2001. (http://www.w3.org/)

[3] Buttner, Andrew, "<Oval SQL='false'>", presentation, The MITRE Corporation, October 2003.

[4] Baker, Mark et al, *XHTML Basic*, W3C Recommendation, December 2000. (http://www.w3.org/)

[5] Bray, Tim et al, *Namespaces in XML*, W3C Recommendation, January 1999. (http://www.w3.org/)

[6] Jones, George, "Introduction to RAT", presentation, Center for Internet Security, October 2003.

[7] Calabrese, Chris, "VulnTrack", presentation at 1st XCCDF Workshop, October 2003.

[8] Casipe, Maria and Schmidt, Charles, "The Open Checklist Interactive Language (OCIL)", The MITRE Corporation, August 2009, (http://scap.nist.gov/specifications/ocil/index.html)

[9] Davis, Mark, "Unicode Regular Expressions", Unicode Technical Recommendation No. 18, version 9, January 2004.

[10] Bartel et al, "XML – Signature Syntax and Processing", W3C Recommendation, February 2002. (http://www.w3.org/)

[11] Marsh, J. and Orchard, D., "XML Inclusions (XInclude) Version 1.0", W3C Candidate Recommendation, April 2004. (http://www.w3.org/)

[12] Hillmann, Diane, "Using Dublin Core", DCMI, August 2003. (http://dublincore.org/)

[13] "Security Configuration Checklists Program for IT Products", NIST Special Publication 800-70, August 2004. (http://checklists.nist.gov/)

[14] Johnston, P. and Powell, A., "Guidelines for Implementing Dublin Core in XML", DCMI, April 2003. (http://dublincore.org/)

[15] Ziring, N. and Wack, J., "Specification for the Extensible Configuration Checklist Description Format (XCCDF)", NIST IR 7188, January 2005.

[16] "OVAL – The Open Vulnerability and Assessment Language", The MITRE Corporation, September 2006. (http://oval.mitre.org/)

[17] Buttner, A., and Ziring, N., "Common Platform Enumeration (CPE) – Name Format and Description, Version 2.0", MITRE Corporation, September 2007. (http://cpe.mitre.org/)

[18] Mell, P., Romanosky, S., and Scarfone, R., "A Complete Guide to the Common Vulnerability Scoring System Version 2.0", FIRST, June 2007. (http://www.first.org/cvss/)

Appendix E. Acronym List

CCE	Common Configuration Enumeration
CIS	Center for Internet Security
COTS	Commercial Off-the-Shelf
CPE	Common Platform Enumeration
CVE	Common Vulnerabilities and Exposures
CVSS	Common Vulnerability Scoring System
DISA	Defense Information Systems Agency
DNS	Domain Name System
DOD	Department of Defense
FIPS	Federal Information Processing Standards
FISMA	Federal Information Security Management Act
FSO	Field Security Office
GOTS	Government Off-the-Shelf
GUI	Graphical User Interface
HIPAA	Health Insurance Portability and Accountability Act
HTML	Hypertext Markup Language
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
ISAP	Information Security Automation Program
IT	Information Technology
ITL	Information Technology Laboratory
MAC	Media Access Control
NIST	National Institute of Standards and Technology
NSA	National Security Agency
NVD	National Vulnerability Database
OCIL	Open Checklist Interactive Language
OMB	Office of Management and Budget
OS	Operating System
OVAL	Open Vulnerability and Assessment Language
SCAP	Security Content Automation Protocol
SP	Special Publication
STIG	Secure Technology Implementation Guide

UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VMS	Vulnerability Management System
W3C	World Wide Web Consortium
XCCDF	Extensible Configuration Checklist Description Format
XHTML	Extensible Hypertext Markup Language
XML	Extensible Markup Language
XSD	XML Schema Definition
XSLT	Extensible Stylesheet Language Transformation