CIS Benchmarks

Center for Internet Security Configuration Assessment Tool CIS-CAT

Policy Customization Guide

v3.0.51

August 14, 2018

Table of Contents

Introduction

The purpose of this document is to describe the components of the CIS-CAT XML Benchmarks and provide instruction on how to perform common customizations to a given CIS-CAT XML Benchmark. While not required, it is recommended that you use a XML editor when working with CIS-CAT XML Benchmarks. All CIS-CAT XML Benchmark are located in the `benchmarks` directory within the CIS-CAT ZIP archive. If you need support customizing CIS-CAT's policies, or have suggestions for how this guide can be improved, please feel free to post a question/comment to the CIS-CAT Discussion forum or send an email to support@cisecurity.org. For a primer in XML basics please review the following tutorials:

- http://www.w3schools.com/xml/xml_tree.asp
- http://www.w3schools.com/xml/xml_syntax.asp
- http://www.w3schools.com/xml/xml_elements.asp
- http://www.w3schools.com/xml/xml_attributes.asp
- http://www.w3schools.com/xml/xml_dtd.asp
- http://www.w3schools.com/xml/xml_validator.asp

Benchmark Composition

The `xccdf:Benchmark` node is the top most level of any XCCDF benchmark, all other nodes will be inside this one. Below is an example of the top part of a benchmark:

```
1  <xccdf:Benchmark xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2  xsi:schemaLocation="http://checklists.nist.gov/xccdf/1.1"
3  xmlns:ecl="http://cisecurity.org/check" xmlns:xi="http://www.w3.org/2001/XInclude"
4  xmlns:xhtml="http://www.w3.org/1999/xhtml"
5  xmlns:xccdf="http://checklists.nist.gov/xccdf/1.1"
6  xmlns:xs="http://www.w3.org/2001/XMLSchema"
7  xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" id="unique_benchmark_id"
8  xml:lang="en">
9    <xccdf:status date="YYYY-MM-DD">accepted</xccdf:status>
10   <xccdf:title>Stub Title</xccdf:title>
11   <xccdf:description>Stub Description</xccdf:description>
12   <xccdf:front-matter>
13     <xhtml:div style="text-align:center;">
14       <xhtml:h1>Stub front matter</xhtml:h1>
15     </xhtml:div>
16   </xccdf:front-matter>
17   <xccdf:rear-matter />
18   <xccdf:reference />
19   <xccdf:version>X.Y.Z.P</xccdf:version>
```

Line 7
This line contains the `id` attribute which specifies the unique ID for a given benchmark.

Line 9
This line specifies the status of the benchmark. Valid values are `accepted` and `draft`. The `date` attribute specifies the date the status was set.

Line 10
The `title` attribute specifies the title of the given benchmark.

Line 11
The `description` element is used for a short description of the benchmark.

Line 12
This line is the start of the `front-matter` node. This node will usually hold a more descriptive version of the benchmark and quite often includes html mark-up for when the XML is used to generate a report. Quite often copyright and company information will be included in this area.

Line 17
This line is the `rear-matter` node which is often not filled out, this will usually hold information that should go at the bottom or end of reports.

Line 18
This line will hold any references used developing the benchmark.

Line 19
This line holds the version number of the benchmark. This `X.Y.Z` portion of the `xccdf:version` node corresponds with the PDF benchmark version number. The `P` portion of the version number is used to differentiate XML benchmark updates.

Profile Composition

Each benchmark can have multiple profiles. Each profile is defined by an `xccdf:Profile` element and contains child nodes that define which `xccdf:Rules` are evaluated when the given `xccdf:Profile` is selected. Below is a stub version of an `xccdf:Profile`:

```
1 <xccdf:Profile id="unique-profile-id">
2     <xccdf:title>Stub Title</xccdf:title>
3     <xccdf:description>Example Description</xccdf:description>
4     <xccdf:select idref="rule-1" selected="true" />
5     <xccdf:select idref="rule-2" selected="true" />
6     <xccdf:select idref="rule-3" selected="false" />
7     <xccdf:refine-value idref="value-1" selector="f00" />
8 </xccdf:Profile>
```

Line 1
The `id` attribute is used as a unique identifier for the given profile.

Line 2
The `xccdf:title` element is used as a human-readable title of this specific profile.

Line 3
The `xccdf:description` element is used as a human-readable description of this specific profile.

Line 4-6
These lines specify what `xccdf:Rule`(s) are run for this profile. The `idref` attribute holds the `id` for a given `xccdf:Rule` and the `selected` attribute specifies if the rule should be used (`true`) or not used (`false`).

Line 7
The `xccdf:refine-value` allows an `xccdf:Profile` to specify which specific value of an `xccdf:Value` node is used when this profile is selected. See Value Composition for more information.

Line 8
The closing `xccdf:Profile` tag.

Group Composition

An `xccdf:Group` is a way to logically create collections of `xccdf:Rules` and `xccdf:Values`. There can be sub-`xccdf:Groups` in a `xccdf:Group`. Each `xccdf:Benchmark` should have at least one `xccdf:Group`. Below is an example `xccdf:Group`:

```
1  <xccdf:Group id="example-grp">
2      <xccdf:title>Example Title</xccdf:title>
3      <xccdf:description>
4        <xhtml:p>Description Placeholder</xhtml:p>
5      </xccdf:description>
6      <xccdf:Value hidden="false" id="ex-val type="string">
7          ...
8      </xccdf:Value>
9      <xccdf:Rule id="example-rul" selected="false">
10         ...
11     </xccdf:Rule>
12 </xccdf:Group>
```

Line 1
The `id` attribute is used as a unique identifier for the given group.

Line 2
The `xccdf:title` element is used as a human-readable title of this specific group.

Line 3
The `xccdf:description` element is used as a human-readable description of this specific group.

Line 6
Holds `xccdf:Value` element that could be used in `xccdf:Rules`.

Line 9
Holds an `xccdf:Rule` element that is used to do the actual checks against a system.

Line 12
Closing `xccdf:Group` tag.

Value Composition

An `xccdf:Value` provides `xccdf:Rule` flexibility when various `xccdf:Profiles` are defined. For example, in a high security profile, the password length requirement may be greater (12) than in a typical enterprise profile (8). In the absence of `xccdf:Value`, two `xccdf:Rules` are required; one for each profile. However, instead of hard coding the password length requirement in an `xccdf:Rule`, the `xccdf:Rule` can reference an `xccdf:Value`. An `xccdf:Value` has a corresponding value and profile selector. When the high security `xccdf:Profile` is selected, the `xccdf:Rule` will use the corresponding value. Below is an example of an `xccdf:Value` node:

```
1  <xccdf:Value id="pw-min-len-val" type="number">
2     <xccdf:description>
3        <xhtml:p>Minimum Password Length</xhtml:p>
4     </xccdf:description>
5     <xccdf:value>8</xccdf:value>
6     <xccdf:value selector="legacy">8</xccdf:value>
7     <xccdf:value selector="ent-desk">8</xccdf:value>
8     <xccdf:value selector="ent-mob">8</xccdf:value>
9     <xccdf:value selector="high">12</xccdf:value>
10 </xccdf:Value>
```

Line 1
The `id` attribute specifies a unique id for this value throughout the benchmark. The `type` attribute is used to specify the data type for this value. Valid `types` are: `string`, `number` and `boolean`.

Line 2
The `xccdf:description` element is used as a human-readable description of this specific value.

Line 5
This is the first `xccdf:value,` and is the default value if no selector is specified in the given profile.

Line 6-9
These `xccdf:values` are specific values that are selected by `xccdf:Profiles`.

Line 10
Closing `xccdf:Value` tag.

Rule Composition

An `xccdf:Rule` is a basic building block of all CIS-CAT XML Benchmarks. Each `xccdf:Rule` corresponds with one or more recommendations made in a CIS Benchmark. Below is an example of an `xccdf:Rule`:

```
1  <xccdf:Rule id="sample xccdf rule 1">
2     <xccdf:title>Sample XCCDF Rule #1</xccdf:title>
3     <xccdf:description>
4         This Rule exemplifies a basic XCCDF Rule
5     </xccdf:description>
6     <xccdf:check system="…">
7        <xccdf:check-content>
8         …
9        </xccdf:check-content>
10    </xccdf:check>
11 </xccdf:Rule>
```

Line 1
Opening tag for `xccdf:Rule`. Every `xccdf:Rule` must have an `id` attribute that is set to a unique value. In the above example, the `id` attribute is set to `sample_xccdf_rule_1`. No other `xccdf:Rule` element may have this `id`. `xccdf:Profiles` are defined by referencing these `id` attributes.

Line 2
The title of the `xccdf:Rule`. The title is typically used to refer to a given `xccdf:Rule` in reports or other documentation. Titles are typically written in a manner that conveys the intent of the Rule, such as:

- Ensure `/etc/password` is owned by `root:root`
- Ensure the `Banner` directive is set in `sshd_config`

Lines 3-5
The description of the `xccdf:Rule`. The `xccdf:description` typically provides information about the configuration subject, such as a registry key or network service. It also provides other information such as security benefit statements, procedures for implementing/auditing the recommended state, and potential impacts associated with implementing the recommended configuration state.

Lines 6-10
The `xccdf:check` can contain one of several check systems, each defined by their own XML schema. The `system` attribute is used to convey which check system is used and dictates the structure of `xccdf:check-content`. CIS-CAT understands the following three check systems:

- Embedded Check Language (ECL) - http://cisecurity.org/check
- Open Vulnerability Assessment Language (OVAL) - http://oval.mitre.org/XMLSchema/oval-definitions-5
- Script Check Engine (SCE) - http://open-scap.org/page/SCE

Line 11
Closing tag for `xccdf:Rule`.

For examples of how to work with Rules, review their associated recipes:

- Changing the Weight of a Rule
- Creating Multi-Check Rules

- Creating a Rule utilizing the Script Check Engine

Embedded Check Language (ECL)

ECL is an XCCDF check system created by the Center for Internet Security and is the primary check system used to express CIS-CAT XML Benchmarks. ECL supports the following generic test types:

- `ecl:platform`
- `ecl:evaluate`
- `ecl:shell-command`
- `ecl:execute`
- `ecl:file-content`
- `ecl:oracle-parameter`
- `ecl:SQL-query`

For examples of how these test types are used, see the ECL Recipes section.

Supported Operands and Data Types

Throughout ECL, there are several places where the check definition requires specifying a data type (`dt`) and operand (`op`). The following is a list of the data types and operands supported by CIS-CAT:

Data Type (dt)

The data type attribute supports the following values:

- `xs:string`
- `xs:float`
- `xs:integer`
- `xs:boolean`

Operand (op)

The operand attribute supports the following values:

- `eq` – equals
- `eqi` – case insensitive equals, available only for the `xs:string` data type
- `ne` – not equal
- `gt` – greater than
- `lt` – less than
- `ge` – greater than or equal
- `le` – less than or equal
- `pm` – pattern match
- `pn` – negated pattern match
- ba – bitwise and
- bo – bitwise or

Regular Expressions

Several CIS-CAT ECL checks, such as `file-content` and `shell-command`, leverage regular expressions. The regular expression syntax and character classes used by CIS-CAT are defined at the following URLs:

- http://www.w3.org/TR/xquery-operators/#regex-syntax
- http://www.w3.org/TR/xmlschema-2/#charcter-classes

Understanding ecl:platform

`ecl:platform` provides the ability for tests to occur only if the current platform matches the `idref` attribute in the `ecl:platform` node.

```
1  <xccdf:check system="http://cisecurity.org/check">
2     <xccdf:check-content>
3          <ecl:platform idref="cpe://any:linux">
4               <ecl:evaluate comment=”test” dt="xs:boolean" op="eq" value="true">
5                 …
6               </ecl:evaluate>
7          </ecl:platform>
8     </xccdf:check-content>
9  </xccdf:check>
```

In the above example, the `ecl:platform` node will limit the execution of the `ecl:evaluate` node to Linux platforms.

ecl:platform Attributes

`ecl:platform` requires a single attribute (`idref`) that contains a CPE value.

ID reference (`idref`)
ID reference supports the following values:

- `cpe://microsoft:windows`
- `cpe://any:linux`
- `cpe://sun:.*`
- `cpe://ibm:aix`
- `cpe://any:unix`

Understanding ecl:evaluate

`ecl:evaluate` provides a common structure for testing the value returned by a specific ECL test.

```
<xccdf:check system="http://cisecurity.org/check">
   <xccdf:check-content>
        <ecl:evaluate comment="calc test" dt="xs:boolean" op="eq" value="true">
             <ecl:FileExists file="C:\Windows\System32\calc.exe" />
        </ecl:evaluate>
   </xccdf:check-content>
</xccdf:check>
```

In the above example, the `ecl:FileExists` test will return a Boolean value of `true` or `false` based on the existence of `C:\Windows\System32\calc.exe`. `ecl:evaluate` compares that return value against the criteria expressed in its data type (`dt`), operand (`op`), and `value` attributes. In this example, `ecl:evaluate` will cast the value returned by `ecl:FileExists` as a Boolean (`xs:boolean`), and test if it equals (`eq`) the value of `true`. If `C:\Windows\System32\calc.exe` exists, this `xccdf:check` will pass.

ecl:evaluate Attributes

`ecl:evaluate` has four attributes: `comment`, data type (`dt`), operand (`op`), and `value` attributes. The following details the possible values for each:

Comment
The contents of the comment attribute are displayed when CIS-CAT generates a report.

Data Type (`dt`)
For a list of data types supported by `ecl:evaluate`, see Data Type (dt).

Operand (`op`)
For a list of operands supported by `ecl:evaluate`, see Operand (op). In addition, `ecl:evaluate` supports the following operands:

- `set_eq` – set equals
- `set_bl` – set does not contain a given value (blacklist)
- `set_wl` – set contains only the given value (whitelist)

Value (`value`)
The value attribute can be set to an arbitrary value.

ecl:evaluate Tests

`ecl:evaluate` supports the following check types:

- Shell-Command
- Execute
- File-Content
- SQL-Query
- Oracle-Parameter
- Win32_RegistryValue
- Win32_RegistryACL
- Win32_FileACL
- Win32_FileAuditPolicy
- Win32_RegistryAuditPolicy
- Win32_PrivilegeAccounts
- Win32_AuditPolicy
- Win32_AccountStatus
- Win32_WMI
- Win32_PasswordPolicy
- FileExists

Understanding ecl:shell-command

`ecl:shell-command` provides a check author with the ability to execute an arbitrary command on the target system, inspect the output and exit status of that command, and make a pass/fail decision based on those values. Review the following recipes for reference implementations for `ecl:shell-command`:

- Pass or Fail a Test Based on a Processes Exit Status
- Pass of Fail a Test Based on the Existence of Absence of Output
- Test if a Given Registry Key/Value Exists
- Test the Access Control List (ACL) of a Given File (DACL)

ecl:shell-command Attributes

Check (**`check`**)
The `check` attribute is optional and is used to specify constraints for the output of the executed command. The following are allowed values:

- `all` – The check will fail if no output (STDOUT) is present.
- `none exist` – The check will fail if output (STDOUT) is present.
- `at least one` – The check will fail if no output (STDOUT) is present.

Success (**`success`**)
The `success` attribute is optional and may be set to either `fail` or `pass`. The success attribute operates on the exit status of the executed command. The following table details the impact the `success` attribute has on the check's result.

success attribute	Process exit status	Check result
pass	0	Pass
pass	Not 0	Fail
fail	0	Fail
fail	Not 0	Pass

Understanding ecl:execute

`ecl:execute` is equivalent to `ecl:shell-command` with one exception; – `ecl:execute` does not invoke the target system's native shell, such as `cmd.exe`, `/bin/sh`, or `/bin/ksh`, as the parent process for running the specified command. Instead it uses Java's ProcessBuilder class to run the command.

Understanding ecl:file-content

`ecl:file-content` provides a check author with the ability to apply a regular expression to a specified file, evaluate expressions based on the regex groups captured by that regular expression, and make a pass/fail decision based on that expression. Review the following recipes for reference implementations for `ecl:file-content`:

- Test if a File Matches a Regular Expression
- Test if a Value in a File is Greater/Less/Equal to a Given Value

ecl:file-content Attributes

Check (**`check`**)
The `check` attribute is optional and is used to specify constraints for the results of retrieving file contents. The following are allowed values:

- `all` – The check will fail if no content is present.
- `none exist` – The check will fail if content is present.
- `at least one` – The check will fail if no content is present.
- `one and only one` – The check will fail if more than one line of content is present.

Understanding ecl:SQL-query

`ecl:SQL-query` can be used in Oracle benchmarks to run queries against a given database to make sure a value exists or does not exist. Each `ecl:SQL-query` node must have a `ecl:query` child node which contains the actual query to be executed.

```
<xccdf:check system="http://cisecurity.org/check">
   <xccdf:check-content>
      <ecl:SQL-query check="none exist">
         <ecl:query>
            select USERNAME from DBA_USERS
            where ACCOUNT_STATUS not like '%LOCKED%' and
            USERNAME in ( 'ANONYMOUS', 'BI', 'CTXSYS')
         </ecl:query>
      </ecl:SQL-query>
   </xccdf:check-content>
</xccdf:check>
```

In the above example the `ecl:SQL-query` node is testing to make sure no rows are returned by setting the `check` attribute to `none exist`. Review the following recipes for reference implementations for `ecl:SQL-query`:

- Test if a Given Query Returns Rows

ecl:SQL-query Attributes

Check (**`check`**)
The `check` attribute is mandatory if not specified then the test is considered informational. The following are allowed values:

- `all` – The check will fail if no results are returned by the query.

- `none exist` – The check will fail if results were returned by the query.

Understanding ecl:oracle-parameter

`ecl:oracle-parameter` is used to check the value of a given configuration parameter. This is done by querying the `V$PARAMETER` table for the value set in `ecl:name` child node.

```
<xccdf:check system="http://cisecurity.org/check">
   <xccdf:check-content>
      <ecl:oracle-parameter check="all">
         <ecl:name>ifile</ecl:name>
         <ecl:value dt="xs:string" op="pm" value=".*"/>
      </ecl:oracle-parameter>
   </xccdf:check-content>
</xccdf:check>
```

In the above example we are checking for the parameter `ifile` which is specified in the `ecl:name` node. We then use the `ecl:value` node to specify the value we expect for this parameter. Review the following recipes for reference implementations for `ecl:oracle-parameter`:

- Test if a Given Oracle Parameter is set a Specific Value
- Test if a Given Oracle Parameter has not been set

ecl:oracle-parameter Attributes

`ecl:oracle-parameter` has only one attribute, `check`, which only has an expected value of `none exist`. If the check attribute is set to something besides `none exist` then the `ecl:value` child-node should be specified.

ecl:value Attributes

Data Type (`dt`)
For a list of data types supported by `ecl:value`, see Data Type (dt).

Operand (`op`)
For a list of operands supported by `ecl:value`, see Operand (op).

Value (`value`)
The value attribute can be set to an arbitrary value.

Understanding ecl:Win32_RegistryValue

`ecl:Win32_RegistryValue` is used to retrieve the value from the Windows registry.

```
<xccdf:check system="http://cisecurity.org/check">
   <xccdf:check-content>
      <ecl:evaluate dt="xs:integer" op="ge" value="1">
         <ecl:Win32_RegistryValue hive="HKEY_LOCAL_MACHINE"
key="SYSTEM\CurrentControlSet\Services\Netlogon\Parameters" name="requiresignorseal" />
      </ecl:evaluate>
   </xccdf:check-content>
</xccdf:check>
```

In example above, `ecl:Win32_RegistryValue` is used to retrieve the value for the registry item `requiresignorseal`. Review the following recipes for reference implementations for `ecl:Win32_RegistryValue`:

- Test if a Given Registry Value is Great/Less/Equal to a Given Value
- Test if a Given Service is Enabled/Disabled

ecl:Win32_RegistryValue Attributes

Hive (**`hive`**)
This attribute is used to specify a registry hive that the value should be pulled from, valid values are: `HKEY_LOCAL_MACHINE`, `HKEY_CLASSES_ROOT`, `HKEY_CURRENT_USER`, `HKEY_USERS` and `HKEY_CURRENT_CONFIG`. This is a required attribute of `ecl:Win32_RegistryValue`.

Key (**`key`**)
This attribute is used to specify the key where the value exists that needs to be retrieved. This is a required attribute of `ecl:Win32_RegistryValue`.

Name (**`name`**)
This attribute is used to specify the name of the registry item that needs to be pulled. This is an optional attribute of `ecl:Win32_RegistryValue`.

Understanding ecl:Win32_RegistryACL

`ecl:Win32_RegistryACL` is used to retrieve the access control list (ACL) for a given key in the Windows registry. This will allow check authors to test the permissions for a specific registry key.

```
<xccdf:check system="http://cisecurity.org/check">
   <xccdf:check-content>
      <ecl:evaluate comment="HKLM\Software" op="eq"
value="D:PAI(A;CI;KA;;;BA)(A;CIIO;KA;;;CO)(A;CI;KA;;;SY)(A;CI;KR;;;BU)">
         <ecl:Win32_RegistryACL hive="HKEY_LOCAL_MACHINE" key="SOFTWARE"/>
      </ecl:evaluate>
   </xccdf:check-content>
</xccdf:check>
```

In the above example the `ecl:Win32_RegistryACL` is retrieving the permissions for `HKLM\Software`. Review the following recipes for reference implementations for `ecl:Win32_RegistryACL`:

- Test the Access Control List (ACL) of a Given Registry Key/Value (DACL)

ecl:Win32_RegistryACL Attributes

Hive (`hive`)
This attribute is used to specify a registry hive that the value should be pulled from, valid values are: `HKEY_LOCAL_MACHINE`, `HKEY_CLASSES_ROOT`, `HKEY_CURRENT_USER`, `HKEY_USERS` and `HKEY_CURRENT_CONFIG`. This is a required attribute of `ecl:Win32_RegistryACL`.

Key (`key`)
This attribute is used to specify the key where the value exists that needs to be retrieved. This is a required attribute of `ecl:Win32_RegistryACL`.

Understanding ecl:Win32_FileACL

`ecl:Win32_FileACL` is used to retrieve the access control list (ACL) for a given file location on the Windows file system. This will allow check authors to test the permissions for directories and files.

```
<xccdf:check system="http://cisecurity.org/check">
   <xccdf:check-content>
      <ecl:evaluate comment="c:\" op="eq"
value="D:(A;OICI;FA;;;BA)(A;OICI;FA;;;SY)(A;OICIIO;GA;;;CO)">
         <ecl:Win32_FileACL path="c:\" />
      </ecl:evaluate>
   </xccdf:check-content>
</xccdf:check>
```

In the above example the `ecl:Win32_FileACL` is retrieving the permissions for `C:\`.

ecl:Win32_FileACL Attributes

Path (`path`)
The path attribute is used to specify the file or directory a check author would want to check permissions against. This is a required attribute of `ecl:Win32_FileACL`.

Understanding ecl: Win32_FileAuditPolicy

`ecl:Win32_FileAuditPolicy` is used to retrieve the audit policy for a specific file or folder on the Windows file system. This allows for a check author to verify if auditing is setup correctly.

```
<xccdf:check system="http://cisecurity.org/check">
   <xccdf:check-content>
      <ecl:evaluate dt="xs:string" op="eq" value="1111000000111111111">
         <ecl:Win32_FileAuditPolicy path="${env:SystemRoot}" userObject="Everyone"
auditType="fail" />
      </ecl:evaluate>
   </xccdf:check-content>
</xccdf:check>
```

In the above example the check is making sure the `%SystemRoot%` has Full Control auditing turned on for the Everyone group. Review the following recipes for reference implementations for `ecl:Win32_FileAuditPolicy`:

- Test the Audit Policy Associated with a Given File (SACL)

ecl:Win32_FileAuditPolicy Attributes

Path (**`path`**)
The file or folder path that needs to have the audit policy checked. This is a required attribute of `ecl:Win32_FileAuditPolicy`.

User Object (**`userObject`**)
The user object that a check author wants to get the audit policy for. This can be a user or a group. This is a required attribute of `ecl:Win32_FileAuditPolicy`.

Audit Type (`auditType`)
The type of audit policy, valid values are: `all`, `success` and `fail`. This is a required attribute of `ecl:Win32_FileAuditPolicy`.

Valid value return values for a SACL are:

- `1111000000111111111` - Full Control
- `1111000000111011111` - Everything except Traverse Folder / Execute File
- `1111000000111111110` - Everything except List Folder / Read Data
- `1111000000101111111` - Everything except Read Attributes
- `1111000000111110111` - Everything except Read Extended Attributes
- `1111000000111111101` - Everything except Create Files / Write Data
- `1111000000111111011` - Everything except Create Folders / Append Data
- `1111000000011111111` - Everything except Write Attributes
- `1111000000111101111` - Everything except Write Extended Attributes
- `1111000000110111111` - Everything except Delete Subfolders and Files
- `1111000000111111111` - Everything except Delete
- `1101000000111111111` - Everything except Read Permissions
- `1011000000111111111` - Everything except Change Permissions
- `1110000000111111111` - Everything except Take Ownership

Understanding ecl: Win32_RegistryAuditPolicy

`ecl:Win32_RegistryAuditPolicy` is used to retrieve the audit policy for a specific key in the Windows registry.

```
<xccdf:check system="http://cisecurity.org/check">
   <xccdf:check-content>
      <ecl:evaluate dt="xs:string" op="eq" value="1111000000111111111">
         <ecl:Win32_RegistryAuditPolicy hive="HKEY_LOCAL_MACHINE" key="Software"
userObject="Everyone" auditType="fail" />
      </ecl:evaluate>
   </xccdf:check-content>
</xccdf:check>
```

In the above example, the check is examining the audit policy assigned to the key `HKLM\Software` registry key to determine if all (Full Control) failed access attempts to this key are logged. Review the following recipes for reference implementations for `ecl:Win32_RegistryAuditPolicy`:

- Test the Audit Policy Associated with a Registry Key/Value (SACL)

ecl:Win32_RegistryAuditPolicy Attributes

Hive (`hive`)
This attribute is used to specify a registry hive that the audit policy should be pulled from, valid values are: `HKEY_LOCAL_MACHINE`, `HKEY_CLASSES_ROOT`, `HKEY_CURRENT_USER`, `HKEY_USERS` and `HKEY_CURRENT_CONFIG`. This is a required attribute of `ecl:Win32_RegistryAuditPolicy`.

Key (`key`)
This attribute is used to specify the key where the audit policy should be pulled from. This is a required attribute of `ecl:Win32_RegistryAuditPolicy`.

User Object (`userObject`)
The user object that a check author wants to get the audit policy for. This can be a user or a group. This is a required attribute of `ecl:Win32_RegistryAuditPolicy`.

Audit Type (`auditType`)
The type of audit policy, valid values are: `all`, `success` and `fail`. This is a required attribute of `ecl:Win32_RegistryAuditPolicy`.

Valid value return values for a SACL are:

- `11110000000000111111` -- Full Control
- `11110000000000111110` -- Everything except Query value
- `11110000000000111101` -- Everything except Set Value
- `11110000000000111011` -- Everything except Create Subkey
- `11110000000000110111` -- Everything except Enumerate Subkey
- `11110000000000101111` -- Everything except Notify
- `11110000000000011111` -- Everything except Create Link
- `11110000000000111111` -- Everything except Delete
- `10110000000000111111` -- Everything except Write DAC
- `11010000000000111111` -- Everything except Read Control
- `01110000000000111111` -- Everything except Write Owner (NOTE: The first zero is truncated)

Understanding ecl: Win32_PrivilegeAccounts

`ecl:Win32_PrivilegeAccounts` is used to retrieve a list of user accounts that possess a specific security privilege. This allows for a check author to verify that only the allowed users and groups are able to perform certain operations.

```
<xccdf:check system="http://cisecurity.org/check">
   <xccdf:check-content>
      <ecl:evaluate dt="xs:string" op="eq" value="Users,Administrators">
         <ecl:Win32_PrivilegeAccounts privilege="senetworklogonright"/>
      </ecl:evaluate>
   </xccdf:check-content>
</xccdf:check>
```

In the above example, the check is making sure only Users and Administrators can login via the network. Review the following recipes for reference implementations for `ecl:Win32_PrivilegeAccounts`:

- Test if a Given Privilege is Limited to a Set of Users or Groups

ecl:Win32_PrivilegeAccounts Attributes

Privilege (`privilege`)
The `privilege` attribute is used to specify which security control a check author wants a user list for. Acceptable values for the `privilege` attribute are:

- `SeAssignPrimaryTokenPrivilege`
- `SeAuditPrivilege`
- `SeBackupPrivilege`
- `SeBatchLogonRight`
- `SeChangeNotifyPrivilege`
- `SeCreateGlobalPrivilege`
- `SeCreatePagefilePrivilege`
- `SeCreatePermanentPrivilege`
- `SeCreateSymbolicLinkPrivilege`
- `SeCreateTokenPrivilege`
- `SeDebugPrivilege`
- `SeDenyBatchLogonRight`
- `SeDenyInteractiveLogonRight`
- `SeDenyNetworkLogonRight`
- `SeDenyRemoteInteractiveLogonRight`
- `SeDenyServiceLogonRight`
- `SeEnableDelegationPrivilege`
- `SeImpersonatePrivilege`
- `SeIncreaseBasePriorityPrivilege`
- `SeIncreaseQuotaPrivilege`
- `SeIncreaseWorkingSetPrivilege`
- `SeInteractiveLogonRight`
- `SeLoadDriverPrivilege`
- `SeLockMemoryPrivilege`
- `SeMachineAccountPrivilege`
- `SeManageVolumePrivilege`
- `SeNetworkLogonRight`
- `SeProfileSingleProcessPrivilege`
- `SeRelabelPrivilege`
- `SeRemoteInteractiveLogonRight`
- `SeRemoteShutdownPrivilege`
- `SeRestorePrivilege`
- `SeSecurityPrivilege`
- `SeServiceLogonRight`
- `SeShutdownPrivilege`
- `SeSyncAgentPrivilege`
- `SeSystemEnvironmentPrivilege`
- `SeSystemProfilePrivilege`
- `SeSystemtimePrivilege`
- `SeTakeOwnershipPrivilege`
- `SeTcbPrivilege`
- `SeTimeZonePrivilege`
- `SeUndockPrivilege`
- `SeUnsolicitedInputPrivilege`

Understanding ecl: Win32_AuditPolicy

`ecl:Win32_AuditPolicy` is used to retrieve what level of auditing is used on a given category of Windows auditing. This allows for a check author to verify that a given category has the correct level of auditing.

```
<xccdf:check system="http://cisecurity.org/check">
   <xccdf:check-content>
      <ecl:evaluate dt="xs:string" op="eq" value="Success">
         <ecl:Win32_AuditPolicy category="AccountLogon"/>
      </ecl:evaluate>
   </xccdf:check-content>
</xccdf:check>
```

In the above example the check is making sure successful logons are recorded. Review the following recipes for reference implementations for `ecl:Win32_AuditPolicy`:

- Test the Audit Policy

ecl:Win32_AuditPolicy Attributes

Category (**`category`**)
Specifies which audit category needs to be checked, valid values are:

- `System`
- `Logon`
- `ObjectAccess`
- `PrivilegeUse`
- `DetailedTracking`
- `PolicyChange`
- `AccountManagement`
- `DirectoryServiceAccess`
- `AccountLogon`

ecl:Win32_AuditPolicy Return Values

The possible values for `ecl:Win32_AuditPolicy` are:

- `No auditing`
- `Success`
- `Failure`
- `Success,Failure`

Understanding ecl: Win32_AccountStatus

`ecl:Win32_AccountStatus` is used to get the account status of a given user. This allows for a check author to verify that certain users are either enabled or disabled.

```
<xccdf:check system="http://cisecurity.org/check">
   <xccdf:check-content>
      <ecl:evaluate dt="xs:string" op="eq" value="disabled">
         <ecl:Win32_AccountStatus account="^S-1-5-[0-9-]+501$" isSid="true"/>
      </ecl:evaluate>
   </xccdf:check-content>
</xccdf:check>
```

In the above example the check is making sure the `Guest` account is `disabled`. Review the following recipes for reference implementations for `ecl:Win32_AccountStatus`:

- Test if a Given User Account is Enable/Disabled

ecl:Win32_AccountStatus Attributes

Account (**`account`**)
This specifies the account status should be checked against, this value can either be a SID or a user name.

Is SID (**`isSid`**)
This specifies if the account value is a SID or not. If not specified it is assumed the value is false.

ecl:Win32_AccountStatus Return Values

The possible values for `ecl:Win32_AccountStatus` are:

- `Enabled`
- `Disabled`
- `Does Not Exist`
- `Unknown`

Understanding ecl:Win32_WMI

`ecl:Win32_WMI` is used to query values from WMI on a given computer. This allows for a check author to verify specific WMI values, it should be noted certain values will only be available on domain-joined machines. An example of this is any value being pulled from `RSOP_SecuritySettingBoolean`. A good tool to test WMI queries is `wmic.exe`.

```
<xccdf:check system="http://cisecurity.org/check">
   <xccdf:check-content>
      <ecl:evaluate dt="xs:string" op="eq" value="False">
         <ecl:Win32_WMI namespace="Root\rsop\computer" wql="SELECT Setting FROM
RSOP_SecuritySettingBoolean WHERE KeyName='LSAAnonymousNameLookup'" />
      </ecl:evaluate>
   </xccdf:check-content>
</xccdf:check>
```

In the above example the check is making sure that anonymous name lookup has been disabled. Review the following recipes for reference implementations for `ecl:Win32_WMI`:

- Perform Arbitrary WMI Tests

ecl:Win32_WMI Attributes

Namespace (**`namespace`**)
The namespace attribute is used to specify what area of WMI the value should be pulled from.

WQL (**`wql`**)
The `wql` attribute holds the actual query that should be used against WMI.

Understanding ecl:Win32_PasswordPolicy

`ecl:Win32_PasswordPolicy` is used to retrieve the settings for a given password related policy.

```
<xccdf:check system="http://cisecurity.org/check">
   <xccdf:check-content>
      <ecl:evaluate dt="xs:integer" op="eq" value="86400">
         <ecl:Win32_PasswordPolicy policyItem="min_passwd_age" />
      </ecl:evaluate>
   </xccdf:check-content>
</xccdf:check>
```

In the above example the check is making sure the minimum password age is set to one day in seconds. Review the following recipes for reference implementations for `ecl:Win32_PasswordPolicy`:

- Test the Password Policy

ecl:Win32_PasswordPolicy Attributes

Policy Item (**`policyItem`**)
The acceptable values for this attribute are:

- `force_logoff` : either a one or zero will be returned. Value states whether the user will be forcibly logged off when an hour limitation hits.
- `min_passwd_len` : the minimum number of characters a password must be
- `max_passwd_age` : amount of time in seconds before a user must change his or her password
- `password_hist_len` : number of previous passwords kept
- `min_passwd_age` : amount of time in seconds that a user must use a password before he or she can change it
- `lockout_observation_window` : number of seconds where an invalid logon will increment the invalid login count
- `lockout_duration` : length of lockout in seconds
- `lockout_threshold` : a number will be returned, representing the number of invalid login attempts before an account is locked out
- `password_complexity` : either a one or zero will be returned. Non-domain-joined computers will always return a one.
- `reversible_encryption` : either a one or zero will be returned. Non-domain-joined computers will always return a zero.

Understanding ecl:FileExists

`ecl:FileExists` is used to check if a given file exists on a system or not. This can be useful when wanting to check for permissions on a file or reading in from a file by first confirming if the file exists. A check author can write if the check should pass or fail if the file does not exist. The `ecl:FileExists` test will either return `true` (if a file exists) or `false` (if a file does not exist).

```
<xccdf:check system="http://cisecurity.org/check">
   <xccdf:check-content>
      <ecl:evaluate comment=”Does Calc.exe Exist” dt="xs:boolean" op="eq"
value="true">
         <ecl:FileExists file="C:\Windows\System32\calc.exe" />
      </ecl:evaluate>
   </xccdf:check-content>
</xccdf:check>
```

In the above example, we do a check to see if calc.exe exists on a computer running the Windows operating system. Review the following recipes for reference implementations for `ecl:FileExists`:

- Test if a File Exists

ecl:FileExists Attributes

`ecl:FileExists` has one attribute, `file` which specifies the file that needs to be checked for existence.

Understanding ecl:Win32_Group

`ecl:Win32_Group` is used to evaluate a given group's member list. This allows for a check author to verify that only certain users exist in a given group.

```
1 <xccdf:check system="http://cisecurity.org/check">
2    <xccdf:check-content>
3       <ecl:evaluate dt="xs:string" op="set_wl" value=" ">
4          <ecl:Win32_Group group="Remote Desktop Users" />
5       </ecl:evaluate>
6    </xccdf:check-content>
7 </xccdf:check>
```

The above example will pass if the Remote Desktop Users group has no members. On line three (3), the `op` attribute is set to `set_wl`, which represents a set white list test. Similarly, the `op` attribute can be set to `set_bl` for a set black list test. The `value` attribute on line three (3) may contain a comma-separated list of users. In this example, the `value` attribute is blank. Therefore, no users are authorized to be member of the Remote Desktop Users group.

ecl:Win32_Group attributes

`ecl:Win32_Group` has only one attribute `group` which contains the name of the group that the user list needs to be retrieved for.

ECL Recipes

General Recipes

Creating a Custom Profile

To create a custom profile, start with the template below:

```
1 <xccdf:Profile id="unique-profile-id">
2     <xccdf:title>Stub Title</xccdf:title>
3     <xccdf:description>Example Description</xccdf:description>
4     <xccdf:select idref="rule-1" selected="true" />
5     <xccdf:select idref="rule-2" selected="true" />
6     <xccdf:select idref="rule-3" selected="false" />
7     <xccdf:refine-value idref="value-1" selector="f00" />
8 </xccdf:Profile>
```

Change the `id` on line 1 to a unique id and then fill out the `xccdf:title and xccdf:description`. These will be used in CIS-CAT when selecting a profile. Finally, figure out the `xccdf:Rules` that need to be run for this profile, and use the `xccdf:select` nodes to specify that the rules should be run. If for some reason an `xccdf:rule` has a `selected` attribute of `true` or the selected attribute is not there, the test should not be run for this profile. Use the `xccdf:select` node and set the `selected` attribute to `false` to stop the check from running. If a specific value should be selected from an `xccdf:Value` node, use the `xccdf:refine-value` to specify the value and its selector.

Adding a Rule to a Profile

To add a rule to a profile, do the following:

- Find the rule that needs to be added to the profile and write down the value from the id attribute
- Go up to the profile the rule needs to be added to and then insert the following XML:

```
<xccdf:select idref="ID-FROM-STEP-1" selected="true" />
```

- Save the file and run it through CIS-CAT to make sure it works.

Removing/Disabling a Rule from/in a Profile

To remove or disable a rule from a profile do the following:

- Find the `xccdf:select` element that has a `idref` attribute with a value of the rule id that needs to be removed/disabled.
- To disable the rule from running, set the `selected` attribute to `false`. To remove the rule from the profile remove that `xccdf:select` element.

Changing the Weight of a Rule

It is possible to set the weight of a rule. Doing this allows a user to make a rule count more than other rules or in some cases make the rule not count towards the total score. To do this, find the rule that needs to be modified and either add or modify the attribute `weight`. The value inside of the `weight` attribute should be a whole positive round number (i.e. 1). If the rule should be informational and have no relationship to the score of the benchmark, setting the weight to zero will make the check run without affecting the score of the benchmark.

Converting "Interactive" Values to "Non-Interactive" Values

Many benchmarks, such as Oracle database benchmarks for example, may contain values which must be input at runtime by the user performing the assessment. These values are called "interactive" because they require user intervention in order to be properly set prior to benchmark execution. Due to this fact, these benchmarks cannot be executed in a scripting environment. These "interactive" values can be modified to "non-interactive" values in order to allow for scripted execution.

Below is an example of the structure of an "interactive" value:

```
<Value id="jdbc.url" type="string" interactive="true" interfaceHint="text">
    <title>Database access JDBC connect string</title>
    <question>What is the database JDBC URL?</question>
    <value/>
    <default>
        jdbc:oracle:thin:system/password@192.168.17.129:1521:CIS
    </default>
</Value>
```

In order to make the value "non-interactive", benchmark editors must perform the following steps:

1. Either remove the `interactive="true"` text or edit the value to `interactive="false"`
2. Remove the `interfaceHint="text"` attribute
3. Remove the `<question>` element
4. Remove the `<default>` element
5. Edit the `<value>` element to the appropriate value to be used in the assessment

Below is an example of the same value from above, made "non-interactive":

```
<Value id="jdbc.url" type="string">
    <title>Database access JDBC connect string</title>
    <value>jdbc:oracle:thin:system/password@192.168.17.129:1521:CIS</value>
</Value>
```

Creating Multi-Check Rules

It is possible to create one `xccdf:Rule` that has multiple checks. To do this, the `xccdf:complex-check` element needs to be used. Below is an example:

```
<xccdf:Rule id="complex-check-example" selected="false">
   ...
   <xccdf:complex-check operator="OR">
      <xccdf:check system="http://cisecurity.org/check">
        ...
      </xccdf:check>
      <xccdf:check system="http://cisecurity.org/check">
         ...
      </xccdf:check>
   </xccdf:complex-check>
</xccdf:Rule>
```

Each `xccdf:complex-check` has two attributes `operator` and `negate`. The `operator` must be specified and it can either be `AND` or `OR`. Doing this will make the results of all the child `xccdf:checks` and `xccdf:complex-checks` be evaluated using Boolean logic. If the `operator` is set to `AND` that means all checks must pass. If the `operator` is set to `OR` then at least one of the checks must pass. When the `negate` attribute is set to `true` then a pass will result in a fail and vice versa.

Creating a Rule utilizing the Script Check Engine

Currently, many administrators use several of their own scripts to make sure their systems follow certain guidelines. These scripts are usually written in Bash, Windows batch files, PowerShell, etc. and are executed as they are. The administrators would like to move to SCAP to allow them to inter-operate and use tools supporting these standards. However they can't afford to make that transition abruptly, it would require them to rework all of their testing scripts at once. In order to ease the transition from scripts to standards-based assessment content, the Script Check Engine concept was developed to allow standards-based content to reference/execute scripts and report on their output. The Script Check Engine was initially developed as part of the OpenSCAP project.

Rules which leverage the use and implementation of the Script Check Engine continue to utilize the standards-based XCCDF constructs. Constructing a Rule utilizing the Script Check Engine is as follows:

```
<xccdf:Rule id="xccdf_sce_rule_1">
   <xccdf:title>Sample XCCDF Rule #1</xccdf:title>
   <xccdf:description>
       This Rule exemplifies a basic XCCDF Rule leveraging the Script Check Engine
   </xccdf:description>
   <xccdf:check system="http://open-scap.org/page/SCE">
      <xccdf:check-import import-name="stdout"/>
      <xccdf:check-content-ref href="ExampleScript.sh"/>
   </xccdf:check>
</xccdf:Rule>
```

To leverage CIS-CAT's implementation of the Script Check Engine, the `xccdf:check` element must have a system attribute set to http://open-scap.org/page/SCE. The optional `xccdf:check-import` element usually references "`stdout`" in order to allow implementations to fetch the script's output and use it in `rule-result` constructs for later examinations. Finally, the `xccdf:check-content-ref` element contains a `href` attribute which indicates the script to be executed. The above example indicates to CIS-CAT to execute the "ExampleScript.sh" shell script, and capture its standard output to use for evaluation of the Rule.
Further information can be found at OpenSCAP's SCE page, located at http://www.open-scap.org/features/other-standards/sce/.

Test if a File Exists

The following `xccdf:Rule` uses the `ecl:FileExists` node to test if the Windows Calculator is present on Windows.

```
1  <xccdf:Rule id="ensure_calculator_exists" selected="false">
2     <xccdf:title>Ensure C:\Windows\System32\calc.exe Exists</xccdf:title>
3     <xccdf:description>
4         The calculator is a very useful tool. This test ensures the calculator is
5         installed.
6     </xccdf:description>
7     <xccdf:check system="http://cisecurity.org/check">
8         <xccdf:check-content>
9             <ecl:evaluate comment="Does Calc.exe Exist" dt="xs:boolean" op="eq" value="true">
10                <ecl:FileExists file="C:\Windows\System32\calc.exe" />
11            </ecl:evaluate>
12        </xccdf:check-content>
13    </xccdf:check>
14 </xccdf:Rule>
```

The above check can be altered to fail if the Windows calculator is present by setting the `value` attribute on line 9 to `false`. Similarly, this test can be altered to test for the presence of a different file by changing the `file` attribute on line 10.

Test if a File Matches a Regular Expression

The following `xccdf:Rule` uses the `ecl:file-content and ecl:line-selection` nodes to test if the `Banner` attribute in `sshd_config` is set to `/etc/issue` on a *nix system.

```
1  <xccdf:Rule id="ensure_banner_set_to_etc_issue_in_sshd_config" selected="false">
2      <xccdf:title>Ensure Banner is set to /etc/issue in sshd_config</xccdf:title>
3      <xccdf:description>
4          Ensure Banner is set to /etc/issue in sshd_config
5      </xccdf:description>
6      <xccdf:check system="http://cisecurity.org/check">
7          <xccdf:check-content>
8              <ecl:file-content check="all" comment="Banner set to /etc/issue">
9                  <ecl:path>/etc/ssh/sshd_config</ecl:path>
10                 <ecl:line-selection op="pm" value="^\s*Banner\s+/etc/issue\s*$"/>
11             </ecl:file-content>
12         </xccdf:check-content>
13     </xccdf:check>
14 </xccdf:Rule>
```

The logic of this test can be inverted in setting the `op` attribute of the `ecl:line-selection` node on line 10 to `pn` – pattern negate. For a list of all operands supported by CIS-CAT, see Operand (op). To cause CIS-CAT to evaluation the contents of a different file, alter the value of the `ecl:path` node. See the section on Regular Expressions for more information CIS-CAT's regular expression support.

Test if a Value in a File is Greater/Less/Equal to a Given Value

The following `xccdf:Rule` uses the `ecl:file-content, ecl:line-selection, and ecl:regex-group` nodes to test if the `LoginGraceTime` attribute in `sshd_config` is less than or equal to `120`.

```
1  <xccdf:Rule id="test_login_grace_time_in_etc_issue_in_sshd_config" selected="false">
2      <xccdf:title>Ensure LoginGraceTime is &lt;= to 120 in sshd_config</xccdf:title>
3      <xccdf:description>
4          Ensure LoginGraceTime is set less than or equal to 120 in sshd_config.
5      </xccdf:description>
6      <xccdf:check system="http://cisecurity.org/check">
7          <xccdf:check-content>
8              <ecl:file-content check="all" comment="LoginGraceTime &lt;= 120 in sshd_config">
9                  <ecl:path>/etc/ssh/sshd_config</ecl:path>
10                 <ecl:line-selection op="pm" value="^\s*LoginGraceTime\s+(\d+)\s*$">
11                     <ecl:regex-group dt="xs:integer" op="le" group="1" value="120" />
12                 </ecl:line-selection>
13             </ecl:file-content>
14         </xccdf:check-content>
15     </xccdf:check>
16 </xccdf:Rule>
```

On line 10, the regular expression defined in the `value` attribute of the `ecl:line-selection` node contains a capture (`(\d+)`), denoted in green. This regular expression has a single capture. The value captured in this regular expression is stored in `regex-group` 1. On line 11, `regex-group` 1 (`group="1"`) is being cast as an integer (`xs:integer`) and compared to the value of `120` using the less-than-or-equal-to operand (`le`). To cause this test to pass if the `LoginGraceTime` is greater-than-or-equal-to `120`, change the `op` attribute to `ge`. For a list of all operands supported by CIS-CAT, see Operand (op). See the section on Regular Expressions for more information CIS-CAT's regular expression support.

Pass or Fail a Test Based on a Processes Exit Status

The following `xccdf:Rule` uses the `ecl:shell-command` node, its `success` attribute, and the UNIX `test(1)` command, to pass a check if `/var/log` is indeed a directory.

```
1  <xccdf:Rule id="confirm_var_log_exists_and_is_a_directory" selected="false">
2      <xccdf:title>Ensure /var/log exists and is a directory.</xccdf:title>
3      <xccdf:description>
4          Ensure /var/log exists and is a directory.
5      </xccdf:description>
6      <xccdf:check system="http://cisecurity.org/check">
7          <xccdf:check-content>
8              <ecl:shell-command comment="/var/log exists and is a directory" success="pass">
9                  test -d /var/log
10             </ecl:shell-command>
11         </xccdf:check-content>
12     </xccdf:check>
13 </xccdf:Rule>
```

On line 8, the `success` attribute of the `ecl:shell-command` node is set to `pass`. By setting this attribute, CIS-CAT will cause the check to pass if the shell command exits with a status code of `0`. If the command exits with a non-zero status code, this `xccdf:Rule` will result in a fail. The man page for the `test(1)` command informs us that `test` will exit with status code of `0` if the expression passed to it is true. To cause this check to fail if `test` returns an exit status code of `0`, set the `success` attribute to `fail`. The `test` command can be replaced with any command or program on your target system.

Pass or Fail a Test Based on the Existence or Absence of Output

The following `xccdf:Rule` uses the `ecl:shell-command` node to verify the permissions of the `/etc/ssh/ssh_config` file.

```
1  <xccdf:Rule id="confirm_ssh_config_permissions" selected="false">
2      <xccdf:title>Ensure /etc/ssh/ssh_config has correct permissions and
3  ownership.</xccdf:title>
4      <xccdf:description>
5          Ensure /etc/ssh/ssh_config has correct permissions and ownership.
6      </xccdf:description>
7      <xccdf:check system="http://cisecurity.org/check">
8          <xccdf:check-content>
9             <ecl:shell-command check="none exist" comment="chown root:root
10 /etc/ssh/ssh_config; chmod 0644 /etc/ssh/ssh_config">
11               <ecl:command>find /etc/ssh/ssh_config -follow -maxdepth 0 ! \( -user root -
12 group root -perm u=rw,g=r,o=r \) -ls</ecl:command>
13               <ecl:line-selection op="pm" value=".+" />
14            </ecl:shell-command>
15         </xccdf:check-content>
16     </xccdf:check>
17 </xccdf:Rule>
```

On line 9, the check attribute of the `ecl:shell-command` node is set to `none exist`. By setting this attribute, CIS-CAT will check to see if any output is returned or not. If any output is returned then this `xccdf:Rule` will result in a fail. This `xccdf:Rule` uses the `ecl:command` to run a command against the OS to get any ownership or permissions that would fail the test, then the `ecl:line-selection` node is used to match any lines that are returned from the command.

Using Environment Variables in Tests

The following `xccdf:Rule` shows the use of environment variables in a test to check for the existence of the `at.exe` file on the Windows platform.

```
1  <xccdf:Rule id="check_at_exe_exists" selected="false">
2      <xccdf:title> Check to see if at.exe exists</xccdf:title>
3      <xccdf:description>
4          Check to see if at.exe exists
5      </xccdf:description>
6      <xccdf:check system="http://cisecurity.org/check">
7          <xccdf:check-content>
8             <ecl:evaluate comment="Check to see if at.exe exists" dt="xs:boolean" op="eq"
9  value="false">
10               <ecl:FileExists file="${env:SystemRoot}\system32\at.exe" />
11            </ecl:evaluate>
12         </xccdf:check-content>
13     </xccdf:check>
14 </xccdf:Rule>
```

On line 10, the file attribute uses `${env:SystemRoot}` as a placeholder for the `SystemRoot` environment variable. To use an environment variable, the format should be `${env:XXXXX}`. In place of XXXXX is the name of the environment variable that should be used in the test.

Microsoft Windows Recipes

Test if a Given Registry Key/Value Exists

The following `xccdf:Rule` checks to see if a given registry value exists or not. This is done by using the `success` parameter to check to see the return value from the `reg query` command.

```
<xccdf:Rule id="check_ie_version_reg_key_exist" selected="false">
    <xccdf:title>Make sure Internet Explorer version key exists.</xccdf:title>
    <xccdf:description>
        Make sure Internet Explorer version key exists.
    </xccdf:description>
    <xccdf:check system="http://cisecurity.org/check">
        <xccdf:check-content>
           <ecl:shell-command check="all" success="pass">
              <ecl:command>reg query "HKLM\Software\Microsoft\Internet Explorer" /v
Version</ecl:command>
           </ecl:shell-command>
        </xccdf:check-content>
    </xccdf:check>
</xccdf:Rule>
```

Test if a Given Registry Value is Great/Less/Equal to a Given Value

The following `xccdf:Rule` uses the ecl:`Win32_RegistryValue` to retrieve the value from the specified location. Then `ecl:evaluate` uses the `op` parameter to specify what the comparison operation should be. In this case, greater than or equal to (`ge`) was selected and the value pulled from the registry is compared to the value specified in the `value` parameter.

```
<xccdf:Rule id="digitally_encrypt_sign_secure_channel" selected="false">
    <xccdf:title>Digitally Encrypt or Sign Secure Channel Data.</xccdf:title>
    <xccdf:description>
        Digitally Encrypt or Sign Secure Channel Data.
    </xccdf:description>
    <xccdf:check system="http://cisecurity.org/check">
       <xccdf:check-content>
          <ecl:evaluate comment="Digitally Encrypt or Sign Secure Channel Data"
dt="xs:integer" op="ge" value="1">
             <ecl:Win32_RegistryValue hive="HKEY_LOCAL_MACHINE"
key="SYSTEM\CurrentControlSet\Services\Netlogon\Parameters" name="requiresignorseal" />
          </ecl:evaluate>
       </xccdf:check-content>
    </xccdf:check>
</xccdf:Rule>
```

Test the Access Control List (ACL) of a Given Registry Key/Value (DACL)

The following `xccdf:Rule` uses the `ecl:Win32_RegistryACL` node to find the ACL for a given registry key. In the example below, `HKLM\Software` is being checked to make sure that there is limited access to it. To figure out the correct DACL for a given registry key, use Powershell and type the following command: `get-acl –path "hklm:\Software…" | format-list`, where `hklm:\Software` is the path to the registry hive and key you want the SDDL for.

```
1  <xccdf:Rule id="hklm-software-rul" selected="false">
2      <xccdf:title>HKLM\Software</xccdf:title>
3      <xccdf:description>
4          HKLM\Software - Administrators Full; System: Full; Creator Owner: Full; Users: Read
5      </xccdf:description>
6      <xccdf:check system="http://cisecurity.org/check">
7          <xccdf:check-content>
8             <ecl:evaluate comment="HKLM\Software" op="eq"
9  value="D:PAI(A;CI;KA;;;BA)(A;CIIO;KA;;;CO)(A;CI;KA;;;SY)(A;CI;KR;;;BU)">
10                <ecl:Win32_RegistryACL hive="HKEY_LOCAL_MACHINE" key="SOFTWARE"/>
11            </ecl:evaluate>
12         </xccdf:check-content>
13     </xccdf:check>
14 </xccdf:Rule>
```

Test the Access Control List (ACL) of a Given File (DACL)

The following `xccdf:Rule` uses `ecl:command` to run the `cacls.exe` command to get the DACL for the `%SystemDrive%`. To find the correct DACL for a customized rule, use `cacls.exe` on Windows 2003 and Windows XP and `icacls.exe` on Windows 2008, Vista and Windows 7. Then paste that DACL into the value area that is highlighted below starting on line 10.

```
1  <xccdf:Rule id="system-drive-rul" selected="false">
2      <xccdf:title>%SystemDrive%</xccdf:title>
3      <xccdf:description>
4          %SystemDrive% - SYSTEM, Administrators, CREATOR OWNER, INTERACTIVE
5      </xccdf:description>
6      <xccdf:check system="http://cisecurity.org/check">
7          <xccdf:check-content>
8             <ecl:shell-command check="none exist">
9                <ecl:command>cacls %SystemDrive%</ecl:command>
10               <ecl:line-selection op="pn" value="((BUILTIN\\Administrators:F)|(NT
11 AUTHORITY\\SYSTEM:F)|(NT AUTHORITY\\INTERACTIVE:R)|(CREATOR OWNER:F)|(^\s*$))" />
12            </ecl:shell-command>
13         </xccdf:check-content>
14     </xccdf:check>
15 </xccdf:Rule>
```

Test the Audit Policy Associated with a Given File (SACL)

The following *xccdf:Rule* uses the *ecl:Win32_FileAuditPolicy* to retrieve the file audit policy on a given file or folder.

```
<xccdf:Rule id="aud-sys-drive-rul" selected="false">
    <xccdf:title>%SystemDrive%</xccdf:title>
    <xccdf:description>
        Ensure /etc/ssh/ssh_config has correct permissions and ownership.
    </xccdf:description>
    <xccdf:check system="http://cisecurity.org/check">
        <xccdf:check-content>
           <ecl:evaluate comment="%SystemDrive% - Everyone: Failures" dt="xs:string" op="eq"
value="1111000000111111111">
              <ecl:Win32_FileAuditPolicy path="${env:SystemRoot}" userObject="Everyone"
auditType="fail" />
           </ecl:evaluate>
        </xccdf:check-content>
    </xccdf:check>
</xccdf:Rule>
```

Test the Audit Policy Associated with a Registry Key/Value (SACL)

The following `xccdf:Rule` uses the `ecl:Win32_RegistryAuditPolicy` to retrieve the file audit policy on a given file or folder.

```
<xccdf:Rule id="aud-hklm-software-rul" selected="false">
    <xccdf:title>HKLM\Software</xccdf:title>
    <xccdf:description>
        HKLM\Software - Everyone: Failures (this key, propagate inheritable permission to
all subkeys)
    </xccdf:description>
    <xccdf:check system="http://cisecurity.org/check">
        <xccdf:check-content>
           <ecl:evaluate comment="HKLM\Software - Everyone: Failures" dt="xs:string" op="eq"
value="1111000000000111111">
              <ecl:Win32_RegistryAuditPolicy hive="HKEY_LOCAL_MACHINE" key="Software"
userObject="Everyone" auditType="fail" />
           </ecl:evaluate>
        </xccdf:check-content>
    </xccdf:check>
</xccdf:Rule>
```

Test if a Given User Account is Enable/Disabled

The following `xccdf:Rule` uses the `ecl:Win32_AccountStatus` to get the status of a specified user. Then the operator (`op`) with a value of `eq` is used to make sure the account is either disabled or enabled. In the example below we are making sure the Guest user is disabled. Note if more than one SID is found with the `account` attribute, only the first account will be returned.

```
<xccdf:Rule id="accts-guest-stat-rul" selected="false">
    <xccdf:title>Accounts: Guest Account Status</xccdf:title>
    <xccdf:description>
        Make sure the Guest account is disabled.
    </xccdf:description>
    <xccdf:check system="http://cisecurity.org/check">
        <xccdf:check-content>
           <ecl:evaluate comment="Check to see if Guest Account is Active" dt="xs:string"
op="eq" value="disabled">
              <ecl:Win32_AccountStatus account="^S-1-5-[0-9-]+501$" isSid="true"/>
           </ecl:evaluate>
        </xccdf:check-content>
    </xccdf:check>
</xccdf:Rule>
```

Test the Audit Policy

The following `xccdf:Rule` uses `ecl:Win32_AuditPolicy` to retrieve the set values for a given audit category. In the example below the `xccdf:Rule` is getting the value for `AccountLogon` audit policy and then making sure it has `Success` and `Failure` set.

```
<xccdf:Rule id="audit-acct-logon-evnt-rul" selected="false">
    <xccdf:title>Audit Account Logon Events</xccdf:title>
    <xccdf:description>
        Ensure we are auditing success and failure for logon events.
    </xccdf:description>
    <xccdf:check system="http://cisecurity.org/check">
        <xccdf:check-content>
           <ecl:evaluate comment="Audit Account Logon Events" dt="xs:string" op="eq"
value="Success,Failure">
              <ecl:Win32_AuditPolicy category="AccountLogon"/>
           </ecl:evaluate>
        </xccdf:check-content>
    </xccdf:check>
</xccdf:Rule>
```

Test if a Given Privilege is Limited to a Set of Users or Groups

The following `xccdf:Rule` uses `ecl:Win32_PrivilegeAccounts` to determine if a given user or group possesses the privilege articulated by the `privilege` attribute. In the example below, the check is looking for the `seNetworkLogonRight` and making sure only Users and Administrators have access. When using the operator (`op`) of `set_wl` or `set_bl`, the `value` attribute can be set to comma-separated values that contain the users or groups that need to be checked.

```
<xccdf:Rule id="acc-from-ntwrk-rul" selected="false">
    <xccdf:title>Access this computer from the network</xccdf:title>
    <xccdf:description>
        Access this computer from the network.
    </xccdf:description>
    <xccdf:check system="http://cisecurity.org/check">
        <xccdf:check-content>
           <ecl:evaluate comment="Access this computer from the network" dt="xs:string"
op="set_wl" value="Users,Administrators">
              <ecl:Win32_PrivilegeAccounts privilege="senetworklogonright"/>
           </ecl:evaluate>
        </xccdf:check-content>
    </xccdf:check>
</xccdf:Rule>
```

Test the Password Policy

The following `xccdf:Rule` uses `ecl:evaluate` and `ecl:Win32_PasswordPolicy` to ensure that the minimum password age is set to the correct value.

```
<xccdf:Rule id="min-pw-age-rul" selected="false">
    <xccdf:title>Minimum Password Age</xccdf:title>
    <xccdf:description>
        Make sure minimum password age is set correctly.
    </xccdf:description>
    <xccdf:check system="http://cisecurity.org/check">
        <xccdf:check-content>
           <ecl:evaluate comment="Minimum Password Age" dt="xs:integer" op="eq"
value="86400">
              <ecl:Win32_PasswordPolicy policyItem="min_passwd_age" />
           </ecl:evaluate>
        </xccdf:check-content>
    </xccdf:check>
</xccdf:Rule>
```

Perform Arbitrary WMI Tests

The following `xccdf:Rule` uses the `ecl:Win32_WMI` to query WMI on a local machine. Using the `namespace` attribute allows a test to specify what area of WMI the query should be executed against, and the `wql` attribute specifies the query. A good tool to test WMI queries is `wmic.exe`.

```
<xccdf:Rule id="net-access-allow-anon-trans-rul" selected="false">
    <xccdf:title>Network Access: Allow Anonymous SID/Name Translation</xccdf:title>
    <xccdf:description>
        Network Access: Allow Anonymous SID/Name Translation.
    </xccdf:description>
    <xccdf:check system="http://cisecurity.org/check">
        <xccdf:check-content>
           <ecl:evaluate comment="Anonymous SID/Name Translation" dt="xs:string" op="eq"
value="False">
              <ecl:Win32_WMI namespace="Root\rsop\computer" wql="SELECT Setting FROM
RSOP_SecuritySettingBoolean WHERE KeyName='LSAAnonymousNameLookup'" />
           </ecl:evaluate>
        </xccdf:check-content>
    </xccdf:check>
</xccdf:Rule>
```

Test if a Given Service is Enabled/Disabled

The following `xccdf:Rule` will check via the `ecl:Win32_RegistryValue` node to see if a service is enabled or disabled. If the value being returned is equal to `4` then a service is disabled. Any other number besides `4` means that the service is not disabled. For instance, the value of `2` means the service is set to start automatically, and a value of `3` means the service is set to start manually.

```
<xccdf:Rule id="confirm_ssh_config_permissions" selected="false">
    <xccdf:title>Ensure /etc/ssh/ssh_config has correct permissions and
ownership.</xccdf:title>
    <xccdf:description>
        Ensure /etc/ssh/ssh_config has correct permissions and ownership.
    </xccdf:description>
    <xccdf:check system="http://cisecurity.org/check">
        <xccdf:check-content>
           <ecl:evaluate comment="Fax Service" dt="xs:integer" op="eq" value="4">
              <ecl:Win32_RegistryValue hive="HKEY_LOCAL_MACHINE"
key="SYSTEM\CurrentControlSet\Services\Fax" name="Start" />
           </ecl:evaluate>
        </xccdf:check-content>
    </xccdf:check>
</xccdf:Rule>
```

Oracle Database Recipes

Test if a Given Query Returns Rows

This `xccdf:Rule` uses the `ecl:SQL-query` and subsequent `ecl:query` to execute a query against a specified Oracle database. The attribute `check` with a value of `all` does a test to see if a given query returns rows; conversely if the check attribute was set to `none exist` the rule would be a test to make sure no rows were returned.

```
1  <xccdf:Rule id="trace_files_public_false" selected="false">
2      <xccdf:title>_trace_files_public= FALSE </xccdf:title>
3      <xccdf:description>
4          _trace_files_public= FALSE
5      </xccdf:description>
6      <xccdf:check system="http://cisecurity.org/check">
7          <xccdf:check-content>
8             <ecl:SQL-query check="all">
9              <ecl:query>
10               select ksppstvl from sys.x$ksppi x, sys.x$ksppcv y where x.indx=y.indx and
11 x.ksppinm='_trace_files_public' and ksppstvl='TRUE'
12             </ecl:query>
13           </ecl:SQL-query>
14         </xccdf:check-content>
15     </xccdf:check>
16 </xccdf:Rule>
```

Test if a Given Oracle Parameter is set a Specific Value

This `xccdf:Rule` uses the `ecl:oracle-parameter` child node to look for the `global_names` parameter which is specified on line 9. The value returned from the database is then compared against the `ecl:value` node using the operator (`op`) attribute and the expected value (`value`) attribute. In this case the value to pass the test is `true`.

```
1  <xccdf:Rule id="global_names_true" selected="false">
2      <xccdf:title>global_names= TRUE</xccdf:title>
3      <xccdf:description>
4          global_names= TRUE
5      </xccdf:description>
6      <xccdf:check system="http://cisecurity.org/check">
7          <xccdf:check-content>
8             <ecl:oracle-parameter check="all">
9                <ecl:name>global_names</ecl:name>
10               <ecl:value dt="xs:boolean" op="eq" value="true"/>
11            </ecl:oracle-parameter>
12         </xccdf:check-content>
13     </xccdf:check>
14 </xccdf:Rule>
```

Test if a Given Oracle Parameter has not been set

This `xccdf:Rule` uses the `ecl:oracle-parameter` child node to see if `trace_enabled` is set. To tell if a parameter is not set, set the `check` attribute to `none exists`. If no value is returned from the database for the given parameter then the test will pass.

```
<xccdf:Rule id="trace_not_enabled" selected="false">
    <xccdf:title>Make sure trace is not enabled</xccdf:title>
    <xccdf:description>
     Make sure trace is not enabled
    </xccdf:description>
    <xccdf:check system="http://cisecurity.org/check">
        <xccdf:check-content>
           <ecl:oracle-parameter check="none exists">
              <ecl:name>trace_enabled</ecl:name>
           </ecl:oracle-parameter>
        </xccdf:check-content>
    </xccdf:check>
</xccdf:Rule>
```

eXtensible Configuration Checklist Description Format (XCCDF)

Rule Results

There are five different XCCDF results that are possible: `fail`, `informational`, `notchecked`, `notselected` and `pass`. Below we will describe what each of these results mean, how these results can occur, and how they will be displayed in a report generated by CIS-CAT.

Fail Result

A `fail` result occurs when a check specified in the ECL does not pass. All `fail` results will be displayed as `fail` from a CIS-CAT report.

Informational Result

An `informational` result is a check that does not have any outcome and is used to gain and/or gather more information about a given system or application. An `informational` result will occur if a given ECL check does not have a `check` and does not have a `success` attribute. An `informational` result will show up under the `informational` column in a CIS-CAT report.

Notchecked Result

A `notchecked` result can mean one of two things: the check defined does not exist within the scanner or there are no checks specified in a given rule. A `notchecked` rule will show up under the informational column and will not be counted against the overall score of a system.

Notselected Result

A `notselected` result occurs when a rule is not specified in a given profile and the selected attribute is set to `false` (the default). A `notselected` rule will show up under the informational column and will not be counted against the overall score of a system.

Pass Result

A `pass` result occurs when all checks succeed against the given check criteria. All `pass` results will be displayed as `pass` from a CIS-CAT report.