D SCAP SURVEY

# Understanding SCAP Through a Simple Use Case

Alexandre D'Hondt
Université Catholique
de Louvain-la-Neuve
Email: alexandre.dhondt@student.uclouvain.be

Hussein Bahmad
Université Catholique
de Louvain-la-Neuve
Email: hussein.bahmad@student.uclouvain.be

***Abstract*—Nowadays, one of the main concerns of IT personnel is security. Indeed, systems composed of various software products must be configured, patched and monitored in compliance with organization's security requirements. As complying with these requirements is a very expensive task, security engineers need a way to automate as much as possible security activities and therefore need to rely on external sources for feeding their tools. That's why the National Institute of Standards and Technology (NIST) developed the Security Content Automation Protocol (SCAP) in order to standardize security information and to fill a gap in interoperability across the security tools. In this review, we try to give an overview of SCAP and more particularly some of its main components and illustrate its purpose with a simple use case focused on system security settings compliance checking.**

**General Terms**: Security, Assessment, Testing, Reporting.
**Keywords**: SCAP, OVAL, XCCDF.

## I. Introduction

Today, complying with organization's security requirements is not a soft job for security engineers due to the large variety of systems and software products. Beyond efficiently configuring, patching and monitoring IT assets, compliance with organization's security requirements has to be guaranteed without overwhelming the IT security personnel. Moreover, these last years, with the emergence of new security vendors, often spending efforts in creating their own nomenclatures for vulnerabilities, various solutions and tools were developed. In order to gain in efficiency, resulting security items have to be shared among industries to save costs and therefore a solution is needed to exchange these items in a standardized fashion. That's why the National Institute of Standards and Technology undertook in the early 2000s to specify the Security Content Automation Protocol in order to standardize security activities and overcome the lack of interoperability between existing solutions. Furthermore, SCAP also provides a simple way to perform security reporting and to demonstrate compliance.

The remainder of this review is structured as follows. In section II, SCAP is presented. It is explained starting with its origin and purpose, then explaining its specification. Two of its main components are presented in a little bit more details and some use cases are mentioned. In section III, a simple practical use case is discussed for illustrating the content of section II through the use of a command line tool from the OpenSCAP standard, especially for applying some basic secure configuration compliance checking tests. Section IV concludes this review of SCAP with some ways ahead.

## II. What is SCAP ?

In fact, SCAP [1] is first of all an attempt from the US Department of Commerce through NIST since the early 2000s to standardize the format by which the security content is communicated with the industry. The main purpose is to give to everybody the same expression for compliance with some main standards such as ISO/IEC 27001 (an international standard specifying requirements in the field of information security management), DoD Directive 8500 (Department of Defense's directive for information assurance requirements) and the Federal Information System Controls Audit Manual (FISCAM). But its purpose is broader and SCAP can also be used for more specialized technical security activities such as digital forensics. Providing such a standardization allows to exchange information between the security experts but also helps to achieve an underlying goal considered as the Holly Grail for the information security management, that is, security automation. [2] [3]

This section first explains the specification of SCAP and its components. It then gives some interesting use cases and success stories. It finally gives a few more details about two of SCAP's components, OVAL and XCCDF, which are the languages targeted by the simple use case described in the next section.

### A. A suite of specifications

*SCAP is a suite of specifications that standardizes the format and nomenclature by which security software products communicate information about software identification, software flaws, and security configurations* [4]. The *SCAP specification*, denoting the representation, must be distinguished from the *SCAP content*, which denotes the data shared across the information security community and hosted on various databases across the world, e.g. the National Vulnerability Database (NVD) [5].

SCAP specification can be dissected in three (disjoint and complementary) categories :

1) **Languages** for checklists and tests specification and reporting,
2) **Enumerations** (dictionaries) of security information,
3) Vulnerability **measurement and scoring** systems.

Note that enumerations, centralized in the SCAP content, ensure that what is defined by the languages can be referred to standardized data.

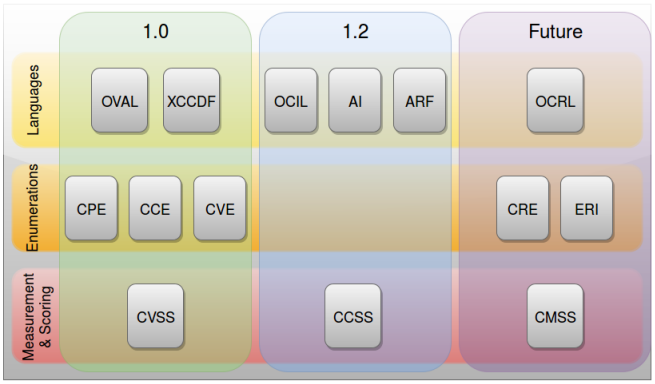


Figure 1. SCAP standard

Figure 1 depicts an overview of the specification, starting on the left vertical frame with the very first version, SCAP 1.0, then representing on the middle frame the additional components of the current version, SCAP 1.2 and then showing on the right frame the components still to be developed in further versions. The three categories encompass the components in the horizontal frames.

Version 1.0 consists of the base components for automating security controls. These components form together a set of independent specifications in XML format for expressing information security knowledge in a standardized way. The languages are aimed to give a meaning to the whole information in the context of the systems to be checked. The enumerations contain the information about platforms, configurations and vulnerabilities. The measurement and scoring systems allow to give a ranking to the results of the tests and to prioritize remediation. The base components are the following :

- **XCCDF**: Extensible Configuration Checklist Description Format is a language for authoring security checklists/ benchmarks and for reporting results of their evaluation.
- **OVAL**: Open Vulnerability and Assessment Language is a language for representing system configuration information, assessing machine state, and reporting assessment results.
- **CPE**: Common Platform Enumeration is a nomenclature and dictionary of hardware, operating systems, and applications.
- **CCE**: Common Configuration Enumeration is a nomenclature and dictionary of software security configurations.
- **CVE**: Common Vulnerability and Exposures is a nomenclature and dictionary of security software flaws.
- **CVSS**: Common Vulnerability Scoring System is a system for measuring the relative severity of software flaw vulnerabilities.

Figure 2 depicts the work flow of SCAP 1.0 components using a tool that implements the specification. Such a tool can be an open-source solution such as OpenSCAP `oscap` [6] or a commercial solution such as Arellia Security Analysis [7]. Concretely, the tool takes an XML file as input written in XCCDF with references to CPE, CCE, CVE and CVSS

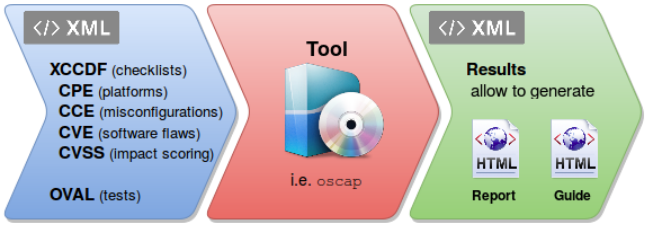


Figure 2. SCAP 1.0 workflow

items and with eventually links to OVAL definitions. It then outputs an XML file from which an HTML report or guide can be generated with the same tool. The resulting HTML report is aimed to highlight the compliance checking results whereas the HTML guide fully describes the compliance checking rules with the related metadata (without performing any test).

Version 1.2 essentially consists of an extension of the languages to provide standardized reporting. It also extends the scoring systems to the configuration items. The additional components are the following :

- **OCIL**: Open Checklist Interactive Language is a language for representing checks that collect information from people or from existing data stores made by other data collection efforts.
- **ARF**: Asset Reporting Format is a format for expressing the transport format of information about assets and the relationships between assets and report.
- **AI**: Asset Identification is a format for uniquely identifying assets based on known identifiers and/or known information about the assets.
- **CCSS**: Common Configuration Scoring System is a system for measuring the relative severity of system security configuration issues.

Future versions of SCAP will extend version 1.2 with some emerging specifications aimed to provide synergy with the already-covered ones in order to achieve more automation tasks such as remediation (note that, at this moment, the data related to remediation exists but is not exploited yet for automation because it's not ready for production). These specifications will soon become part of the NIST validation program and should be made available after this process. Some emerging specifications are the following :

- **OCRL**: Open Checklist Reporting Language is a language for writing XML definitions gathering systems information in standardized reports for policy compliance.
- **CRE**: Common Remediation Enumeration is a nomenclature for remediation activities.
- **ERI**: Extended Remediation Information is a collection of information in addition to the CRE to make this support organizations' remediation activities.
- **CMSS**: Common Misuse Scoring System a system for measuring the characteristics of software feature misuse vulnerabilities.

### B. A solution for many usages

SCAP was developed to address a large part of the information security management concerns in an automated way. These concerns relate to the main activities providing control on the information technologies. The interested reader can refer to the CIS Security Controls to get a complete list [8]. The main use cases with their related security control questions are the following :

- **Asset inventory**: What are the assets in my network ?
- **Vulnerability assessment**: Is my system vulnerable to any exploit ?
- **Patch management**: Can my system cope with the latest flaw findings ?
- **Configuration management**: Is my system configured based on the best practices ?
- **Policy compliance**: Can my system comply with the given policies ?

Up to now, several studies have been led in the field of automating these use cases. From building a service-oriented architecture for vulnerability assessment systems [9] or modelling network attacks using vulnerability information [10] to developing configuration check tools [11] [12] or designing security in mobile devices [13], SCAP is more and more adopted. Moreover, as information technologies security becomes the banner call for most organizations, audit and monitoring [14] [15] increasingly rely on SCAP.

### C. A composition of XCCDF and OVAL

As stated in section II-A, XCCDF is a specification language that allows us to define checklists. Together, these form a benchmark for a given platform. Each checklist consists of a set of rules logically grouped. XCCDF syntax is based on XML and is then structured according to an XML schema. An XCCDF rule is a high-level definition which will be translated to a check on the related system (identified by its platform identifier referring to a CPE). In fact, the rules are not specified directly inside the XML file, instead they point to other XML documents referred as OVAL definition files.

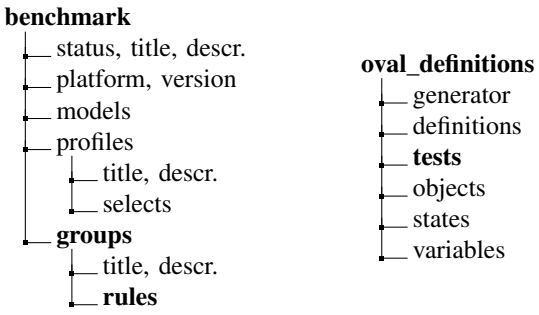


Figure 3. XCCDF (left) and OVAL (right) XML trees (not exhaustive).

In Figure 3, the left tree represents the main part of the XML tree for an XCCDF benchmark. We notice the *benchmark* root with multiple *group* child nodes with themselves multiple *rule* child nodes. These last ones may contain references to OVAL definitions.

XCCDF can also deal with Script Check Engine (SCE), a small and simple check engine that allow to use old home-made check script content written in another language and these old scripts can be mixed with OVAL definitions as illustrated in Figure 4. The currently supported languages are Bash, Python, Perl and Ruby.

```
<Group id="python-script">
   <title xml:lang="en-US">Check with python ...</title>
   <description xml:lang="en-US">...</description>
   <Rule id="rule-id" selected="true">
      <title xml:lang="en-US">Script using Python</title>
      <check system="http://open-scap.org/page/SCE">
        <check-content-ref href="python_script.py"/>
      </check>
   </Rule>
</Group>
<Group id="oval-test">
   <title xml:lang="en-US">Check with Oval ...</title>
   <description xml:lang="en-US">...</description>
   <Rule id="rule2-id" selected="true">
     <title xml:lang="en-US">Script using Oval</title>
     <check system="http://open-scap.org/page/SCE">
       <check-content-ref href="test-oval.xml"/>
     </check>
   </Rule>
</Group>
```

Figure 4. Mix of an OVAL definition and a Python script in an XCCDF file.

Note that XCCDF rules definition also includes fix items which are simple and straightforward remediations to misconfiguration against the tested configuration items. The aware reader may have noticed that remediations should be handled in a future version of SCAP as stated in Figure 1, that is, CRE should provide a specific definition in a dictionary of remediations and could be referred in the XCCDF rules instead of hard-coding them.

As also stated in section II-A, OVAL is a specification language that allows us to define tests. An OVAL definition consists of multiple tests referring to objects (i.e. a file name, a registry key) and states (i.e. file's md5 hash, registry key's value).

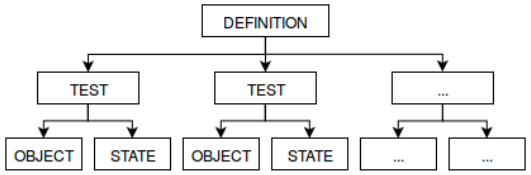


Figure 5. OVAL definition logical structure.

In Figure 3, the right tree represents the main part of the XML tree for an OVAL definition. We notice the *oval_definitions* root with multiple *definition*, *test*, *object* and *state* child nodes. These are logically structured as depicted in Figure 5, that is, they point to each other to provide a meaning that matches this representation.

Both XCCDF and OVAL have their own online repositories [5] [16] with shared SCAP content in order to provide community-developed benchmarks, vulnerability, compliance, inventory and patch definitions for a set of supported operating systems.

## III. A Simple Use Case

This section presents a simple use case for the sake of better understanding SCAP. First, we state the problem and the scope of the use case. Then, we discuss a solution without SCAP and its caveats. Afterwards, we discuss a solution with SCAP, explaining the workflow of its involved components. Finally, we argue the added value of such a solution.

### A. Problem setting

In information security management, meeting the security requirements is a challenging task, especially for checking compliance with organization's policies. For example, passwords can be required to respect some criteria in order to ensure a minimum security level on the systems. Given a chosen platform, Ubuntu Server 14.04, the objective of the present use case is to fulfil the security requirements of the following password policy :

1) Password Files
   a) Only `root` can own `/etc/shadow`
   b) Only `root` can own `/etc/gshadow`
   c) Only `root` can own `/etc/passwd`
   d) Only `root` group can own `/etc/passwd`
   e) `/etc/passwd` must have permissions `0644`
2) Proper Storage & Password Existence
   a) Prevent login accounts with an empty password
   b) All account passwords must be shadowed (hashes)
3) Expiration Parameters
   a) Password minimum age must be set
   b) Password maiximum age must be set
   c) Password warning age must be set

### B. Manual solution

We can imagine that such assessments without SCAP should be done manually. Using other resources, i.e. a company may hire employees to fulfil these needs. However, there could exist some internal implementations with specific languages, enumerations and metrics in order to automate the testing activities, requiring just a few experts to manage the automated execution. This is precisely a case mentioned in the introduction of this review, showing a lack of interoperability that SCAP can overcome.

More concretely, let us suppose that an organisation has to check whether the current installed infrastructure complies with some internal policies such as to forbid an empty password. This verification is a piece of cake for an administrator, checking on each system whether there exists any instance of the `nullok` option in `/etc/pam.d/system-auth` which prevents login with empty passwords. However, the complexity of assessment will increase rapidly depending on the number of rules to test and the number of hosts to be assessed. Sometimes, it requires some knowledge and security experts to conduct assessments of organization policies. This can quickly become too costly.

### C. Automated solution with SCAP

As mentioned in section II-B, an interesting use case of SCAP is the policy compliance checking. In order to illustrate this, we test the previously-defined password policy against a checklist in XCCDF language with an open-source tool. Thus, in order to check for empty passwords like in section III-B, we just need to download the corresponding rule from the NVD and to install a tool on the target system. The check can even be performed remotely from a central system by using a specific tool via the SSH protocol (i.e. with `oscap-ssh`). In the present case, we test a local solution in a virtual machine.

The settings of this experiment are the following :

- Platform : Ubuntu Server 14.04
- Tool : `oscap` (from the OpenSCAP project)
- Checklist : XCCDF file available from a project [17]

In order to execute the test with the given tool, we adapt the password policy into an XCCDF file downloaded from the NVD by defining a profile containing the rules referring to OVAL definitions also downloaded, then forming a benchmark for the given platform. Afterwards, we execute on the targeted system the following simple command :

```
oscap xccdf eval
    --profile xccdf_password_policy
    --cpe ubuntu-cpe.xml
    --oval-results
    --results results.xml
    --report results.html
    ubuntu-xccdf.xml
```

The anatomy of this command is the following :

- `oscap xccdf eval ubuntu-xccdf.xml` tests the system against the given benchmark
- `--profile` selects the right profile, that is, this of Ubuntu Server 14.04, defined for our specific policy
- `--cpe` selects the right platform enumeration (for providing metadata)
- `--oval-results` enables inclusion of additional OVAL information in the XCCDF report
- `--results` generates the given XML file with the XCCDF results
- `--report` generates a human-readable HTML report with very detailed information

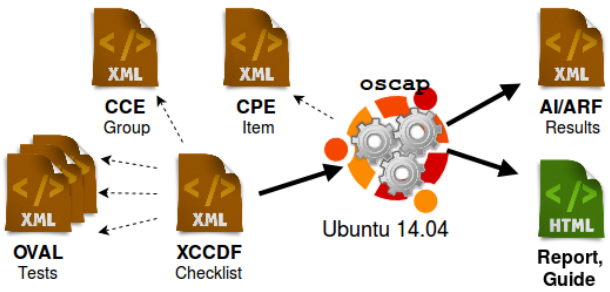


Figure 6. SCAP workflow applied to the simple use case.

Figure 6 presents the workflow for the simple use case. XCCDF checks the CPE item against some OVAL tests based on CCE items from a password policy rules, then generating an XML file with the results and the corresponding HTML report.

The hardest work is certainly to create our own benchmark, requiring some knowledge concerning the standard and the controls related to the tested password policy. Fortunately, SCAP content from some main available sources such as the NVD or Database Exploits contain a broad database of benchmarks and the one we need can be easily found and tailored to the policy.

| Title | Result |
|---|---|
| Verify User Who Owns shadow File | pass |
| Verify User Who Owns gshadow File | pass |
| Verify User Who Owns passwd File | pass |
| Verify Group Who Owns passwd File | pass |
| Verify Permissions on passwd File | pass |
| Prevent Log In to Accounts With Empty Password | pass |
| Verify All Account Password Hashes are Shadowed | pass |
| Set Password Minimum Age | fail |
| Set Password Maximum Age | fail |
| Set Password Warning Age | pass |

Figure 7. List of password policy rules with pass/fail status.

The final report contains a lot of metadata about the configuration items and their potential misconfiguration and also the remediations that can be performed. A nice feature of SCAP is that its ARF specification provides a standardized reporting structure that can be used to generate a user-friendly report in the form of a guidance documentation. Figure 7 shows the summary list of the check status for each tested rule, as an introduction of the generated report. Each rule in the shown table links to a section with detailed information.

### D. Added value

From the explanations of subsections III-B and III-C on the simple use case, we can see that it's very simple to perform a quick compliance checking test on a common system relying on some public resources thanks to SCAP, far more easier than by applying a manual solution.

By using a standard such as SCAP, security information sharing and security automation are made convenient. As current SCAP content gathers a large quantity of knowledge, security experts can simply behave as content consumers and thus spare a significant amount of time.

By contrast, security experts can also contribute as content producers and dealing with the XML schemas quickly becomes complex and costly. So, SCAP content is of course dependent on the available manpower for feeding the repositories and the content is not necessarily complete. However, SCAP community enjoy many contributing organizations making the SCAP content a reliable source for security automation.

As a proof of efficiency, we can mention a few already-released SCAP-validated products from main software companies such as Microsoft SCAP extensions for Microsoft System Center Configuration Manager 3.0, Tenable Security Center 5 or also Qualys SCAP Auditor 1.2 [18].

## IV. Conclusion

We presented SCAP and its specification as a composition of several categories of sub-specifications. We discussed the various use cases it covers through a couple of references to research projects. We gave a few more details about how its two main languages look like and how to formulate checklists and tests. We showed a simple use case of compliance checking based on a simple password policy to illustrate how easy to use SCAP is in comparison to a manual solution. We commented its added value to assess its usefulness and concluded it's a very interesting tool for many usages.

SCAP is an evolutionary effort of the information security community. It will continue to progress from its current version 1.2 with emerging specifications such as CRE for automating remediations. As its usage seems to continuously grow, SCAP will certainly remain a reliable and efficient source for supporting security experts' work.

## References


[1] NIST, *The Technical Specification for the Security Content Automation Protocol (SCAP): SCAP Version 1.2*, Sept 2011, rev. 2.

[2] P. Kampanakis, "Security automation and threat information-sharing options," *Security Privacy, IEEE*, vol. 12, no. 5, pp. 42–51, Sept 2014.

[3] R. Montesino and S. Fenz, "Automation possibilities in information security management," in *Intelligence and Security Informatics Conference (EISIC), 2011 European*, Sept 2011, pp. 259–262.

[4] S. Radack and R. Kuhn, "Managing security: The security content automation protocol," *IT Professional*, vol. 13, no. 1, pp. 9–11, Jan 2011.

[5] National vulnerability database. [Online]. Available: https://nvd.nist.gov/

[6] OpenSCAP. Oscap tool. [Online]. Available: http://www.open-scap.org/

[7] Arellia. Security analysis solution. [Online]. Available: http://www.arellia.com/products/security-analysis-solution/

[8] Cis critical security controls. [Online]. Available: http://www.cisecurity.org/critical-controls/

[9] A. Nakamura, "Towards unified vulnerability assessment with open data," in *Computer Software and Applications Conference Workshops (COMPSACW), 2013 IEEE 37th Annual*, July 2013, pp. 248–253.

[10] K. Ingols, M. Chu, R. Lippmann, S. Webster, and S. Boyer, "Modeling modern network attacks and countermeasures using attack graphs," in *Computer Security Applications Conference, 2009. ACSAC '09. Annual*, Dec 2009, pp. 117–126.

[11] E. Al-Shaer and M. Alsaleh, "Configchecker: A tool for comprehensive security configuration analytics," in *Configuration Analytics and Automation (SAFECONFIG), 2011 4th Symposium*, Oct 2011, pp. 1–2.

[12] M. Alsaleh and E. Al-Shaer, "Scap based configuration analytics for comprehensive compliance checking," in *Configuration Analytics and Automation (SAFECONFIG), 2011 4th Symposium*, Oct 2011, pp. 1–8.

[13] C.-L. Kuo and C.-H. Yang, "Security design for configuration management of android devices," in *Computer Software and Applications Conference (COMPSAC), 2015 IEEE 39th Annual*, vol. 3, July 2015, pp. 249–254.

[14] M. Aslam, C. Gehrmann, and M. Björkman, "Continuous security evaluation and auditing of remote platforms by combining trusted computing and security automation techniques," in *Proceedings of the 6th International Conference on Security of Information and Networks*, ser. SIN '13. New York, NY, USA: ACM, 2013, pp. 136–143. [Online]. Available: http://doi.acm.org/10.1145/2523514.2523537

[15] R. Savola and P. Heinonen, "Security-measurability-enhancing mechanisms for a distributed adaptive security monitoring system," in *Emerging Security Information Systems and Technologies (SECURWARE), 2010 Fourth International Conference on*, July 2010, pp. 25–34.

[16] Oval repository. [Online]. Available: https://oval.mitre.org/repository/

[17] GovReady. Ubuntuscap project. [Online]. Available: https://github.com/GovReady/ubuntu-scap

[18] Scap validated products. [Online]. Available: https://nvd.nist.gov/SCAP-Validated-Tools/