Generation of OVAL Checks for Security-Configuration Assessments

Bachelor's Thesis

Supervisor: Prof. Dr. Alexander Pretschner
Advisor: Patrick Stöckle (patrick.stoeckle@tum.de)
Contact@Siemens: Dr. Bernd Grobauer (bernd.grobauer@siemens.com)

Starting date: immediately

Keywords: Security, Configuration Management, Security Assessment

Department of Informatics
Informatics 4
Software & Systems Engineering
Prof. Dr. Alexander Pretschner

Boltzmannstraße 3
85748 Garching bei München

Website: www4.in.tum.de

Context

The *Open Vulnerability and Assessment Language OVAL*[1] is an XML-based language for expressing security-configuration checks. OVAL is used mainly within security-configuration guides (also called baselines or benchmarks) expressed in in the *Security-Automation Content Protocol (SCAP)*[2]. SCAP/OVAL is a well-established standard implemented by many configuration scan solutions, e.g., CIS-CAT Pro[3]; a publicly available implementation of a SCAP processor is, e.g. , RedHat's OpenScap Workbench and the `oscap` command line tool.[4]

Unfortunately, OVAL is very complicated to write: authoring even a straightforward check such as comparing a Windows registry value against a desired value requires substantial training and extreme care in writing the convoluted and cross-reference-heavy XML syntax of OVAL. Thus, OVAL is virtually unusable as a tool for security experts to author and maintain security checks.

Example

In a security-configuration guide authored by the *Center for Internet Security (CIS)* for a Windows system, we have the rule with the title *Ensure 'Turn off notifications network usage' is set to 'Enabled'*. The remediation of the rule states that we should configure the setting *Computer Configuration \ Policies \ Administrative Templates \ Start Menu and Taskbar \ Turn off notifications network usage* to *Enabled*. We have a corresponding check for this rule defined as an OVAL check, c.f. Listing 1.

```
1 <definition class="compliance" id="oval:def:25962900" version="1">
2   <metadata>
3     <title>(L2) Ensure 'Turn off notifications network usage' is set to 'Enabled'</title>
4     <description>(L2) Ensure 'Turn off notifications network usage' is set to
      ↪  'Enabled'</description>
5   </metadata>
6     <criteria negate="false" operator="AND">
7     <criterion negate="false" test_ref="oval:tst:25962900"/>
8   </criteria>
9 </definition>
```

Listing 1: Example of check definition.

The check refers in line 7 to the real test for the registry. This test has the id `tst:25962900` and is depicted in Listing 2.

```
1 <win:registry_test check="all" check_existence="at_least_one_exists" comment="Ensure
  ↪  'NoCloudApplicationNotification' is 'Windows: Registry Value' to '1'"
  ↪  id="oval:tst:25962900" version="1">
2   <win:object object_ref="oval:obj:25962900"/>
3   <win:state state_ref="oval:ste:25962900"/>
4 </win:registry_test>
```

Listing 2: Example of registry test definition.

As we can see here, the test is referring to an object with the id `obj:25962900` and its desired state with the id `ste:25962900`. The creators of OVAL decided to model objects and their state separately from each other. Their motivation was that we, as authors of security-configuration checks, could reuse objects in our checks. A tool can assess the value of an object

Application:
Please apply via email to patrick.stoeckle@tum.de. Your email should explain your interest in the topic and contain your current transcript of records and a CV. The most promising candidates will be invited for an informal interview. Upon mutual agreement, the thesis can be sponsored by the Siemens AG.

[1] https://oval.cisecurity.org/
[2] https://csrc.nist.gov/projects/security-content-automation-protocol
[3] https://www.cisecurity.org/cybersecurity-tools/cis-cat-pro/
[4] https://www.open-scap.org/tools/openscap-base/

once and evaluate different states against it. We can see the object with the id `obj:25962900` in Listing 3 and the state with the id `ste:25962900` in Listing 4.

```
1 <win:registry_object comment="Ensure 'NoCloudApplicationNotification' is 'Windows: Registry
  ↪ Value' to '1'" id="oval:obj:25962900" version="1">
2   <win:hive>HKEY_LOCAL_MACHINE</win:hive>
3   <win:key operation="case insensitive
    ↪ equals">SOFTWARE\Policies\Microsoft\Windows\CurrentVersion\PushNotifications</win:key>
4   <win:name>NoCloudApplicationNotification</win:name>
5 </win:registry_object>
```

Listing 3: Example of an object definition.

In the object definition, we find all the necessary information to get the value of this object. Here, we can see a registry's hive, key, and name in a Windows operating system.

```
1 <win:registry_state comment="Ensure 'NoCloudApplicationNotification' is 'Windows: Registry
  ↪ Value' to '1'" id="oval:ste:25962900" version="1">
2   <win:type>reg_dword</win:type>
3   <win:value datatype="int" operation="equals">1</win:value>
4 </win:registry_state>
```

Listing 4: Example of a state definition.

In the state definition, we define the desired state of the object. In our example, we define that our object should have the type `DWORD` and the value should be equal to `1`.

The definitions seem to be straight forward, but there are a couple of problems. First, what we presented here next to each other is scattered throughout the whole OVAL file. The check's definition starts in line 3060, the test's in line 7011, the object's in line 9333, and the state's in line 11763. Thus, it is a tedious job to gather all information needed to understand an OVAL check, but even more challenging to write a check without mistakes. Second, the mapping from the group policy editor's description, as presented above, to the hive, key, and name of the registry is not trivial. Until very recently[1], there was no tool publicly available that could translate a description into those values.

```
1 - system: org.scapolite.implementation.win_gpo
2   ui_path: Computer Configuration\Policies\Administrative Templates\Start Menu
3       and Taskbar\Notifications\Turn off notifications network usage
4   value: Enabled
5 - system: org.scapolite.implementation.windows_registry
6   config: Computer
7   registry_key: SOFTWARE\Policies\Microsoft\Windows\CurrentVersion\PushNotifications
8   value_name: NoCloudApplicationNotification
9   action: DWORD:1
```

Listing 5: Example of a YAML-based specification for the automatic implementation.

Our article describes how we extract the essential parts of the rules to implement them automatically. We used a YAML-based syntax to present the extracted values as well as the derived values. Listing 5 presents an example of extracted and derived values. The focus of the paper was on the automatic implementation of rules, not on the automatic checking. Nevertheless, we can interpret this simple example also as a specification of a check. In this case, the semantics of this simple check would be that the value of the setting at the path *Computer Configuration ... network usage* equals `Enabled`. We can translate these semantics to the derived values. Here, the semantics of our simple check are the following: the value of the registry at the hive `Computer` and key `SOFTWARE \ ...` and with value name `NoCloudApplicationNotification` should be of type `DWORD` and have the value `1`. Given those semantics that the `value` field in line 4 and the `action` field in line 9 mark the desired values, we could use this syntax to express simple checks and translate them to OVAL. Of course, this small example only covers Windows-related settings for which this translation is easily possible. As we also want to specify checks for Linux-based systems concisely, we have to come up with more sophisticated solutions. Furthermore, the example only covers the trivial $=$ case. Nevertheless, this example gives a good impression of what a new definition of checks could look like and what it should be capable to do.

Department of Informatics
Informatics 4
Software & Systems Engineering
Prof. Dr. Alexander Pretschner

Boltzmannstraße 3
85748 Garching bei München

Website: `www4.in.tum.de`

Application:
Please apply via email to `patrick.stoeckle@tum.de`. Your email should explain your interest in the topic and contain your current transcript of records and a CV. The most promising candidates will be invited for an informal interview. Upon mutual agreement, the thesis can be sponsored by the Siemens AG.

Third, SCAP's focus is on the automatic checking, not on the automatic application of security-configuration guides. Therefore, if we created our OVAL checks, we would have to apply the rules manually on a test system, execute the OVAL checks, and verify that the checks are compliant. Only with this manual application of the rules we can get at least some confidence that the checks are doing what they are supposed to do.

Forth, this check is relatively simple as we only test for equality. Other checks could use more complicated operators like $<$, $\in$, $\subset$, etc. We have to express these operators in the corresponding OVAL syntax to make them automatically executable.

Goal

In an ongoing co-operation between TUM and Siemens, we could make significant advances in the use of SCAP via a YAML-based format called *Scapolite*. With Scapolite, we could ease the management of security-configuration guides, and we built the tooling to apply them automatically. In this context, this thesis aims to extend this work by generating OVAL checks from intuitive and concise YAML-specifications of security checks. The first step would be to design this concise structure for defining those checks. This structure should contain every information needed to generate OVAL checks from it. Furthermore, this structure should cover all use cases of checks in Windows-related guides. We will use the benchmark of Windows-related 12 guides used in [1], one or two Siemens-internal guides, and Linux-related guides from the CIS to identify all use cases. The structure should be easily understandable. Additionally, it would be preferable to extend the natural language processing presented in Stöckle et al. [1] to create the specifications for the OVAL checks in an automated or a semi-automated way. In the end, we will also use the security-configuration guides of Siemens and the CIS to evaluate the percentage of correctly generated OVAL checks.

Working Plan

1. Familiarize with SCAP and OVAL
2. Familiarize with the Scapolite format and tooling (in Python 3.x) co-created by Siemens and TUM
3. Analyze publicly available SCAP/OVAL documents and develop a YAML-based check-language for the most common and most relevant security checks
4. Write a plugin for the Scapolite tooling, which transforms the developed check language into OVAL
5. Evaluate the created tooling by
   - comparing results of generated OVAL exists with existing OVAL checks.
   - applying existing guides of Siemens and the CIS to test instances and checking them with the generated OVAL checks and state-of-the-art scanners like CIS-CAT Pro

Deliverables

- Source code of the implementation.
- Technical report with comprehensive documentation of the implementation, i.e. design decision, architecture description, API description and usage instructions.
- Final thesis report written in conformance with TUM guidelines.

References

[1] Stöckle, P., Grobauer, B., and Pretschner, A. Automated Implementation of Windows-related Security-Configuration Guides. ASE '20, IEEE Press.

Department of Informatics
Informatics 4
Software & Systems Engineering
Prof. Dr. Alexander Pretschner

Boltzmannstraße 3
85748 Garching bei München

Website: www4.in.tum.de

Application:
Please apply via email to patrick.stoeckle@tum.de. Your email should explain your interest in the topic and contain your current transcript of records and a CV. The most promising candidates will be invited for an informal interview. Upon mutual agreement, the thesis can be sponsored by the Siemens AG.