# Design of an Improved Scapolite Check Mechanism

Bachelor's Thesis


**Supervisor:** Prof. Dr. Alexander Pretschner
**Advisor:** Patrick Stöckle
**Email:** {alexander.pretschner, patrick.stoeckle}@tum.de
**Phone:** +49 (89) 289 - 17314

**Starting date:** immediately


Fakultät für Informatik
Lehrstuhl 4
Software & Systems Engineering
Prof. Dr. Alexander Pretschner

Boltzmannstraße 3
85748 Garching bei München

Tel: +49 (89) 289 - 17314
https://www4.in.tum.de


## Context

We at the Chair of Software and Systems Engineering (I4) are developing the Scapolite hardening-framework in cooperation with an industry partner. The goal of this Scapolite framework is to make the specification and management of security-configuration guidelines easier and automate the implementation and the check of the different security-configuration rules. Therefore, we extract the values we need to implement and to check the current rule from the human-readable text of the security-configurations. The Scapolite framework is implemented in Python, but we have also created a PowerShell Library that implements and checks the rules on the Windows machines. Recently, we demonstrated how one can automate almost the whole hardening process using the configuration of a Windows system as a proof of concept. Currently, we are deriving both the automation for the implementation and for the check from one single value. This covers most of the security settings but for some settings the person who created the security-guideline specified

- **Range** a range of values which are valid, e.g., $x \in [0..10]$.
- **Set** a set of values which are valid, e.g., $x \in \{Disabled, Not\ Configured\}$.
- **Negations** anything but a specific value, e.g., $x \neq Enabled$ or $x \notin \{Disabled, Not\ Configured\}$.

In the current implementation of the Scapolite framework, this is simply ignored and only one value is chosen. This leads to *false negatives* when applying the automated check mechanisms, e.g.: A security-configuration rule specifies that the minimal password length should be at least 12. Thus, in the current statue, the value 12 is extracted and the automation checks if the minimal password length on the system under test is set to 12. If minimal password length is set to 13 on the system under test, the rule is marked as *non-compliant*, although the value 13 is valid according to the specification of the rule. In the course of this thesis, this behavior should be improved.

## Goal

The goal of this thesis is to develop an approach specifying checks more precisely in the Scapolite framework. Hereby, concepts from the OCL [3, 5], first-order logic, or OVAL [4] may be useful and could be reused. The new approach to specify checks in the Scapolite language should include the current definition for which the value to implement and the value to check is the same. Furthermore, the different cases defined in the security-configuration guidelines like ranges, sets or negations should be possible. After the approach is defined, an improved extraction process has to be implemented. In addition, the automation of the checks has to be implemented. The evaluation of this thesis could be conducted in a case study using an existing security-configuration guideline, e.g., the IASE Windows Server 2016 guideline [2] or the Windows Server 2016 guideline published by the CIS[1]. In the end, the degree of rules, which were specified using any aforementioned construct and for which the values can be extracted and their check can be automated, should be as high as possible. On the other side, the new check mechanism should reduce the amount of the aforementioned *false negatives*.

## Working Plan

1. Familiarize with the Scapolite framework
2. Collect possible cases to handle in existing security-configuration guidelines
3. Define concept to specify different check conditions within the Scapolite framework
4. Implement the extraction process
5. Implement the enhanced check automation mechanism
6. Evaluate the results on an existing security-configuration guideline

## References


[1] Center for Internet Security (CIS). https://www.cisecurity.org/. Accessed: 2019-01-29.

[2] IASE Microsoft Windows Server MS DC 2016 STIG Benchmark. Available from https://iasecontent.disa.mil/stigs/zip/U_MS_Windows_Server_2016_V1R7_STIG_SCAP_1-2_Benchmark.zip. Accessed: 2019-01-22.

[3] Object Constraint Language. https://www.omg.org/spec/OCL/2.4. Accessed: 2019-02-20.

[4] J. Baker, M. Hansbury, and D. Haynes. Open Vulnerability and Assessment Language, MITRE CORPORATION. Available from http://oval.mitre.org/language/version5.11/oval-language-specification_12-18-2014.pdf.

[5] Jordi Cabot and Martin Gogolla. *Object Constraint Language (OCL): A Definitive Guide*, pages 58–90. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-30982-3. doi: 10.1007/978-3-642-30982-3_3. URL https://doi.org/10.1007/978-3-642-30982-3_3.



Fakultät für Informatik
Lehrstuhl 4
Software & Systems Engineering
Prof. Dr. Alexander Pretschner

Boltzmannstraße 3
85748 Garching bei München

Tel: +49 (89) 289 - 17314
https://www4.in.tum.de