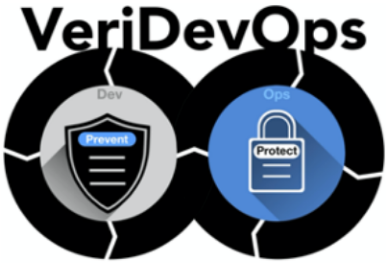

VeriDevOps

Automated Protection and Prevention to Meet Security

Requirements in DevOps Environments

D2.7. Patterns Catalogue

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957212. This document reflects only the author's view and the European Commission is not responsible for any use that may be made of the information it contains.

Contract number:	957212
Project acronym:	VeriDevOps
Project title:	VeriDevOps: Automated Protection and Prevention to Meet Security Requirements in DevOps Environments
DELIVERY DATE	30/03/2022
Author(s):	Andrey Sadovykh (SOFT), Aleksandr Naumchev (SOFT), Eduard Enoiu (MDU), Cristina (MDU)
Partners contributed:	SOFT, MDU
Date:	30/03/2022
Version	1.0
Revision:	1
Abstract:	This report accomplies the software deliverable 2.7 that includes the list implemented patterns for security requirements. In particular, this report provides instructions on obtaining and instantiating the patterns.
Status:	Submitted

DOCUMENT REVISION LOG

VERSION	REVISION	DATE	DESCRIPTION	AUTHOR
	01	01/03/2022	Draft	SOFT, MDH
	02	17/03/2022	SOFT: added contributions MDH: added contributions	SOFT, MDH
	03	30/03/2022	Review	MI, MDH
	04	31/03/2022	Submitted	

TABLE OF CONTENTS

Executive Abstract

VeriDevOps project [1] automates the analysis of security requirements in the DevOps context. One of the major goals is to automate specification, analysis and verification of security requirements as, for example, indicated in standards such as IEC 62443 [2] and Security Technical Implementation Guides [3].

The current deliverable briefly presents the User's Guide for the RQCODE prototype [4], [5] follows Seamless Object-Oriented Requirements [6], [7] paradigm in Java and outlines the currently implemented requirements patterns. This User's Guide gives instructions on installation and getting started. Moreover, this document overviews the tools by MDH including NALABS, GWT, PROPAS and TEARS.

The current deliverable is a follow step towards the automation and will be further improved in agreement with the industrial users.

1. Introduction

In VeriDevOps, ***Work package 2 - Automated generation of security requirements*** investigates automatic extraction, formalization and verification of the security requirements from natural language requirements, vulnerability databases and standards. The resulting information is used as input for WP4 - Prevention at development and WP3 - Reactive Protection at Operations.

The second task of WP2, ***T2.1: Security Formal Modelling*** explores methods for modeling the security requirements in the form of observer timed automata and TCTL queries. It also investigates the suitability of such formalisms for security requirements and proposes certain extensions. The task also investigates the definition of domain specific languages to make the formalism more expressive and easy to use by domain experts. Finally the task analyses the body of knowledge for specifics of security properties formal specification for a set of patterns. The current *software* deliverable, ***D2.7: Patterns catalogue*** presents the current state of catalogue of patterns for formal specification of security properties. This catalogue is presented from the end-user perspective.

The main contribution of this deliverable is RQCODE - Requirements as Code repository [5]. The document provides a brief User's Guide helping to install and get started with the RQCODE security patterns. Section 2 is the getting started overview. Section 3 gives guidelines for troubleshooting.

1.1. RQCODE Reminder

The RQCODE approach has been introduced in VeriDevOps deliverable D2.2 [8]. RQCODE stands for "Requirements as Code" approach that derives from Seamless Object-Oriented Requirements defined in [6]. Requirements are represented as classes in Java. This representation may offer many important advantages:

- Class may incorporate various notations for requirements such as textual form or LTL.
- Class may include verification and validation means - that way a lightweight formalisation of a requirement may be achieved.
- Requirements in a form of class may be extended and instantiated with various parameters providing a means for massive reuse.

In order to implement the approach one should implement *Requirement* interface from *rqcode.concepts* package. In addition, *Checkable* and *Enforceable* interfaces are available to augment requirements with verification means.

1.2. Patterns, Anti-Patterns and Smells Specifications

The pattern specification patterns and anti-patterns have been introduced in VeriDevOps deliverable D2.2 [8]. The advantage of using these patterns and detecting anti-patterns relates to reuse, reduction of ambiguity, and improvement of comprehensibility. In the next subsections we will detail how to access the patterns and anti-patterns catalogue related to the following methods:

- Anti-patterns and smells in natural language requirements (NALABS)
- Given-when-then patterns (GWT)
- Ontology-based patterns (RESA) and Real-time specification patterns (PROPAS)
- TEARS patterns (TEARS)

2. Getting Started

2.1. RQCODE

As it was mentioned previously, please refer to [8] for an RQCODE example. A typical RQCODE will look like:

```
/**
 * Removing the Network Information Service (NIS) package decreases the risk
of the accidental (or intentional) activation of NIS or NIS+ services.
 *
```

```
 *
https://www.stigviewer.com/stig/canonical_ubuntu_18.04_lts/2021-06-16/finding/
V-219157
 */
public class V_219157 implements Checkable {
    @objid ("18fc05f1-f81e-4a17-a5b0-68cc87d1aff4")
     private UbuntuPackagePattern _package = new UbuntuPackagePattern("nis",
false);

    public CheckStatus check() {
        return _package.check();
    }

    public String toString() {
        return _package.toString();
    }

}
```

In this example, UbuntuPackagePattern security requirement is applied for the package *nis*. The behaviour is reused for the all set of such security requirements.

2.1.1. Accessing the RQCODE Catalogue

The RQCODE source code and patterns repository is accessible at:

https://github.com/anaumchev/VDO-Patterns/

To contribute you may request access from andrey.sadovykh@softeam.fr

2.1.2. Structure of the repository

The current structure of the repository is depicted in the following figures:

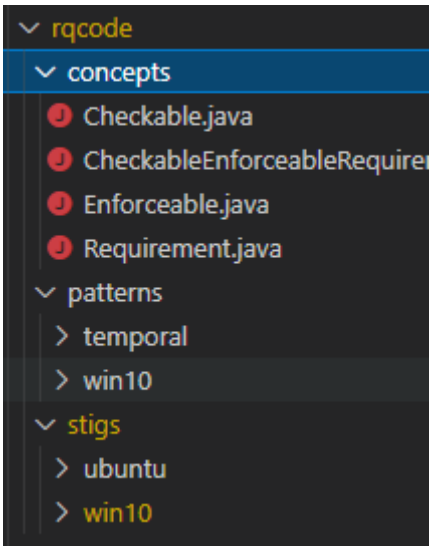

Figure 1. Structure of the RQCODE repository.

The *rqcode* package contains *concepts*, *patterns* and *stigs* subpackages.

- *concepts* include the major interfaces that RQCODE classes implement.
- *patterns.temporal* contain the major temporal patterns such as GlobalUniversality.
- *patterns.windows* include specific patterns extracted from the Window 10 related STIGs.
- *stigs.ubuntu* and *stigs.win10* are examples of the concrete security requirements from the STIG repository.

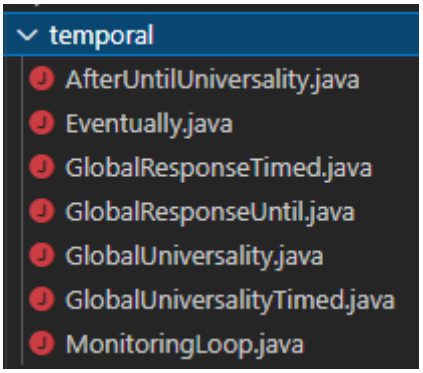

Figure 2. Temporal RQCODE patterns

2.1.3. Executing an RQCODE example

One may check the RQCODE execution examples in:

- https://github.com/anaumchev/VDO-Patterns/blob/master/src/Main.java for temporal properties requirements combined with Windows 10 STIGs requirements.
- https://github.com/anaumchev/VDO-Patterns/blob/master/src/rqcode/stigs/win10/Windows10SecurityTechnicalImplementationGuide.java for Windows 10 STIGs only.

- https://github.com/anaumchev/VDO-Patterns/blob/master/src/rqcode/stigs/ubuntu/Main.java for Ubuntu STIGs only.

```java
import com.modeliosoft.modelio.javadesigner.annotations.objid;
import rqcode.patterns.temporal.GlobalResponseTimed;
import rqcode.patterns.temporal.GlobalUniversalityTimed;
import rqcode.stigs.win10.V_63449;
import rqcode.stigs.win10.V_63467;

@objid ("0246ff65-342a-4bc3-bc46-4d5acfa86b02")
public class Main {
    @objid ("2d10f699-25fa-4cad-9f0a-6878681e9529")
    public static void main(String[] args) {
        V_63449 v_63449 = new V_63449();
        GlobalUniversalityTimed globalUniversality_V_63449 = new GlobalUniversalityTimed(v_63449, Integer.MAX_VALUE);
        V_63467 v_63467 = new V_63467();
        GlobalResponseTimed globalResponseTimed_V_63449_V_63447 =
                new GlobalResponseTimed(v_63449, v_63467, Integer.MAX_VALUE);
        GlobalResponseTimed globalResponseTimedComposed =
                new GlobalResponseTimed(globalUniversality_V_63449, globalResponseTimed_V_63449_V_63447, Integer.MAX_VALUE);

        System.out.println(globalResponseTimedComposed);
        System.out.println(globalResponseTimedComposed.TCTL());
        System.out.println (globalUniversality_V_63449.check());
    }

}
```

Figure. Example of RQCODE instantiation.

2.2. Patterns, Anti-Patterns and Smells Specifications (MDH)

In this section we cover the set of requirement patterns (guides and templates used for creating and modifying requirements) as well as the anti-patterns (commonly occurring requirement problems or class of problems that generate negative consequences during development) used in VeriDevOps. The anti-patterns focus on specifications expressed in natural language (i.e., NALABS) while patterns target the specification of requirements in structured and semi-structured language related to formal verification and software testing (i.e., TEARS, RESA, PROPAS, GWT).

2.2.1. Accessing the MDH Catalogues

NALABS. We use the idea of smells and anti-patterns to specifications expressed in natural language, defining a set of specifications bad smells. We developed a tool called NALABS (NAtural LAnguage Bad Smells) available on https://github.com/eduardenoiu/NALABS and used it for automatically checking specifications. Based on several natural language smells observed in previous studies, we established a set of indicators for requirement flaws and defined dictionary-based metrics to automatically detect these smells in natural language artefacts.

TEARS. These patterns can be found at https://bitbucket.org/danielFlemstrom/napkin. This is an interactive integrated development and analysis environment for TEARS patterns. More details on how to access the catalogue and use the TEARS patterns can be found on https://bitbucket.org/danielFlemstrom/napkin/src/main/. TEARS was introduced as a specification syntax for independent guarded assertions (G/As).

RESA and PROPAS specification patterns can be found in Github at https://github.com/eduardenoiu/ReSA-Tool-VeriDevOps. The patterns used in PROPAS can also be found in an externally-maintained catalog project at https://github.com/hub-se/PSP-UPPAAL[1]. The PROPAS tool provides the necessary means for generating formal system specifications (CTL, TCTL) based on Specification Patterns. The resulting formalized requirements can then be used in various model checkers such as UPPAAL for algorithmic formal verification. RESA is focusing on requirements specification in constrained natural language in the domain of automotive systems development. It renders natural language terms (words, phrases), and syntax, which gives readability of requirements specification. Moreover, it uses boilerplates to structure the construction of requirements specification.

GWT patterns can be used using the TIGER tool found in the following repository: https://github.com/MuhammadNoumanZafar/TestScriptGeneration. The Given-When-Then was proposed by Dan North as part of behavior-driven development (BDD). Given-When-Then (GWT) is a semi-structured approach to writing requirements and test specifications more closely related to testing and test cases.

2.2.2. Structure of the Repositories

The **NALABS** anti-patterns catalogue can be found directly under the metrics folder: https://github.com/eduardenoiu/NALABS/tree/master/RCM/Metrics. In this section, we describe the structure of the repository related to the quality of natural language specifications, how we created the set of smells and the automatic measurement of these smells using specific dictionaries:

- ConjunctionMetric.cs
- ContinuancesMetric.cs
- ICountMetric.cs
- ImperativesMetric.cs
- NVMetric.cs
- OptionalityMetric.cs
- References2.cs
- ReferencesMetric.cs
- SubjectivityMetric.cs

[1]

- WeaknessMetric.cs

These relate to the following anti-patterns and smells:

- **Vagueness** is a common problematic property when it comes to understanding requirements and requirements complexity.
- **Referenceability.** This is usually an indication of nesting in the requirements documents or a need for additional reading in order to understand the requirement that contains references
- **Optionality.** Optional words are giving the developers a latitude of interpretations to satisfy the specified statements and their use is usually not recommended in requirements documentation.
- **Subjectivity.** Subjectivity metric is measuring personal opinions or feelings in sentences.
- **Weakness** is a metric that counts words and phrases that may introduce uncertainty into requirements statements by leaving room for multiple interpretations.
- **Readability.** Automated Readability Index (ARI) is calculated using W S + 9 × SW, where WS is the average number of words per sentence and SW is the average number of letters per word.
- **Over-Complexity Metrics.** Measuring the size of the requirement was used in a couple of studies. It can be defined in many different ways such as the total number of characters, number of words, paragraphs and lines of text.

The **TEARS** global project settings[2] assumes that there has to be a session directory that contains logs and TEARS g/a's. Note that the back-end server needs to write to this directory, so it typically resides on the same level as the git repo (so it gets mounted properly when starting the container). The structure of that directory looks as follows:

```
session
├── GA
│   ├── TEARS requirements.txt
├── generated
│   ├── ANALYSIS_overview.html
├── log
│   ├── logs
│   └── Expert-Sessions
│       ├── LOGDATA.TXT
│       └── LOGDATA.TXT
├── main_definitions.ga
└── req
```

[2] https://bitbucket.org/danielFlemstrom/napkin/src/main/

For **PROPAS** patterns, one can access the already developed catalogue[3] documented in the wiki https://github.com/hub-se/PSP-UPPAAL/wiki and one can see the structure of these patterns and their implementation as observer automata: https://github.com/hub-se/PSP-UPPAAL/tree/master/Tool/observer_templates.

In the **GWT** test generation repository[4], the 'JsonReading' class contains the implementation detail to read the abstract test cases and a customised method for deserialization of information contained in a Json file in a List of 'DataModel' class objects. 'Signal' class contains the model for storing information about the signals. 'xmlReader' takes the information about signals in xml format and stores the relevant information in a List of Signals. Mapping Rules are defined in the 'TestGenerator' class which are used to concretize the abstract test cases generate the scripts using 'ScriptCreator' class. Program.cs is the main executor class.

2.2.3. Using the MDH Catalogues for Patterns, Anti-Patterns and Smells Specifications

NALABS is composed of two main components: the GUI as the main program executable and the metrics used as proxy for bad smells. The latest release of the NALABS executable can be downloaded from GitHub[5] on the releases page. Alternatively, it can be built from source code. You can use different methods to build an application: the Visual Studio IDE and the MSBuild command-line tools. Add the package Microsoft.Office.Interop.Excel using the NuGet Package Manager. First change some settings. Choose Edit/Settings menu tab. In the Excel view you should choose the REQ ID and Text column in the requirement excel document. To open a requirement excel file choose the File/Open menu tab.

For accessing **TEARS[6] patterns**, you need to have the NAPKIN tool installed. For this, you need to have Docker installed on your local machine. We decided to use a dockerized development environment since it is quite a lot of work to get all dependencies to work. The dependencies are specificed in two places: in the <napkin>/Docker/pyproject.toml and <napkin>/client/package.json. You can start the development servers: In <napkin>/client: npm run dev.

For **RESA**, one needs to create an empty project and create a new file with an extension .resa (generic specification), .vl - for east-adl vehicle-level specification, .al - for east-adl analysis-level specification and .dl - for east-adl design-level specification.

[3] This catalog is the basis for PROPAS patterns which will be updated during the project.
[4] https://github.com/MuhammadNoumanZafar/TestScriptGeneration
[5] https://github.com/eduardenoiu/NALABS/releases
[6] For more detailed instructions you can find a tutorial video on Zenodo: https://zenodo.org/record/4662060

For **GWT** patterns used through Graphwalker, an interface has been developed to provide input files i.e Model file in Json or GraphML (as GraphWalker supports these two formats of model for generating abstract test cases) We have also defined mapping rules to concretize the abstract test cases. A customised class can be added to generate test scripts of your own choice.

3. Troubleshooting

3.1. RQCODE

3.1.1. Known Issues

- The current set of SITG patterns is not exhaustive. This set is continuously updated. The list of SITGs to be encoded is prioritised based on the VeriDevOps case study partners suggestions.

3.1.2. Issues Reporting

Report issues through the GitHub mechanisms.

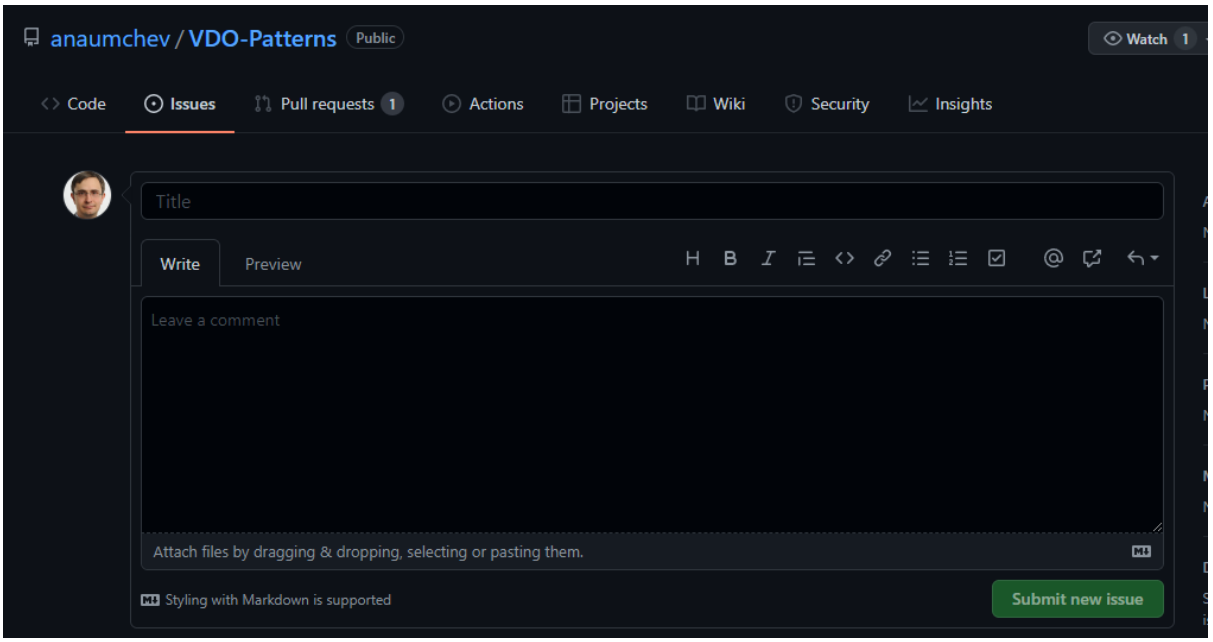

Figure 4. GitHub Issue Reporting

3.1.3. Suggesting New Features for RQCODEs

One may suggest new features through GitHub's mechanism of labelling issues. Use the “enhancement” label for suggesting new features.

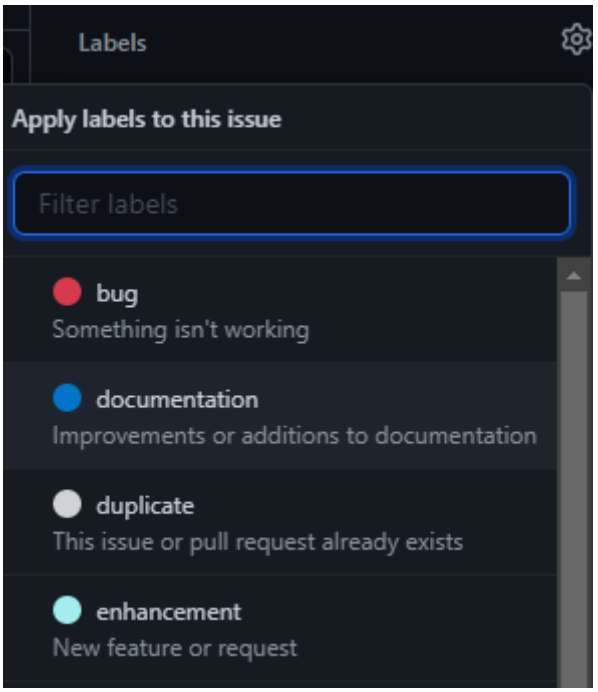

Figure 5. Feature request with GitHub.

4. Conclusions

This report provides a getting started information that accompanies the catalogue for security requirements patterns and their instatiations. In particular the tools by SOFT and MDH are overviewed. These tools will be further extended based on the feedback from the case studies.

References

[1] A. Sadovykh *et al.*, 'VeriDevOps: Automated Protection and Prevention to Meet Security Requirements in DevOps', in *2021 Design, Automation Test in Europe Conference Exhibition (DATE)*, Feb. 2021, pp. 1330–1333. doi: 10.23919/DATE51398.2021.9474185.

[2] IEEE/ISO/IEC, 'International Standard - Systems and software engineering – Life cycle processes – Requirements engineering', IEEE Standards Association, 29148–2018, 2018.

[3] 'Security Technical Implementation Guide (STIG) Complete List', *STIG Viewer | Unified Compliance Framework®*. https://www.stigviewer.com/stigs (accessed Jan. 14, 2022).

[4] K. Ismaeel, A. Naumchev, A. Sadovykh, D. Truscan, E. P. Enoiu, and C. Seceleanu, 'Security

Requirements as Code: Example from VeriDevOps Project', in *2021 IEEE 29th International Requirements Engineering Conference Workshops (REW)*, Sep. 2021, pp. 357–363. doi: 10.1109/REW53955.2021.00063.

[5] A. Naumchev, A. Sadovykh, and K. Ismaeel, *RQCODE GitHub Repository*. 2022. Accessed: Jan. 14, 2022. [Online]. Available: https://github.com/anaumchev/VDO-Patterns

[6] A. Naumchev, 'Seamless Object-Oriented Requirements', in *2019 International Multi-Conference on Engineering, Computer and Information Sciences (SIBIRCON)*, 2019, pp. 0743–0748.

[7] A. Naumchev, 'Exigences orientées objets dans un cycle de vie continu', phdthesis, Université de Toulouse, Université Toulouse III-Paul Sabatier, 2019.

[8] 'VeriDevOps. D2.2 Specification of patterns for security requirements'. [Online]. Available: https://www.veridevops.eu/download/18.6e35a01117f44a73834e8/1646125616627/D2.2.pdf

Annex 1 - RQCODE Reference Specification

Hereafter we provide reference documentation for the RQCODE package.

Package "rqcode.concepts"

from Package **rqcode**

Stereotypes: JavaPackage

Contains the major RQCODE concepts.

Name	Summary
Enforceable	
Checkable	

Table 2 Owned Interfaces of Package "concepts"

Name	Summary
Requirement	
CheckableEnforceableRequirement	

Table 3 Owned Classes of Package "concepts"

Interface "Enforceable"

*from Package rqcode.***concepts**

Stereotypes: JavaInterface

Implementations of this interface are requirements that can be enforced on the hosting environment programmatically through the enforce function.

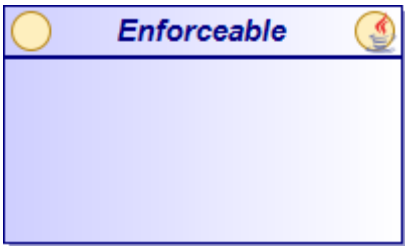

Figure 3 Enforceable (ClassStructureDiagramTemplate)

Name	Description
EnforcementStatus enforce ()	Modifies the hosting environment to satisfy the requirement.

Table 4 Operations of Interface "Enforceable"

Name	Values	Description
EnforcementStatus	SUCCESS FAILURE INCOMPLETE	

Table 5 Owned Enumerations of Interface "Enforceable"

Interface "Checkable"

*from Package rqcode.***concepts**

Stereotypes: JavaInterface

Implementations of this interface are requirements that can be checked programmatically through the check function.

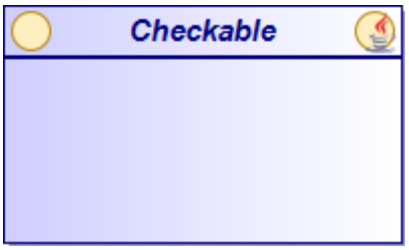

Figure 4 Checkable (ClassStructureDiagramTemplate)

Name	Description
CheckStatus check ()	Checks whether the current environment satisfies the requirement of not.

Table 6 Operations of Interface "Checkable"

Name	Values	Description
CheckStatus	PASS FAIL INCOMPLETE	

Table 7 Owned Enumerations of Interface "Checkable"

Class "Requirement"

*from Package rqcode.***concepts**

Stereotypes: JavaClass

This class is a direct mapping of the structure of STIG findings as presented in stigviewer.com. All the member names are self-explanatory.

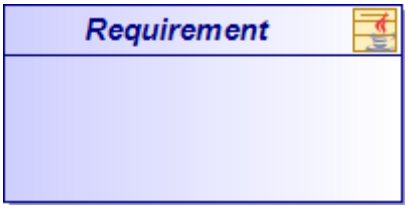

Figure 5 Requirement (ClassStructureDiagramTemplate)

Name	Description
string findingID ()	
string version ()	
string ruleID ()	
string iAControls ()	
string severity ()	
string description ()	
string sTIG ()	
string date ()	
string checkTextCode ()	
string checkText ()	
string fixTextCode ()	
string fixText ()	
string toString ()	A crude parsing of the finding (requirement) specification into a document. Hopefully in the future we would parse it as HTML.

Table 8 Operations of Class "Requirement"

Class "CheckableEnforceableRequirement"

*from Package rqcode.***concepts**

Implements: CheckableImplements: Enforceable

Inherits from: Requirement

Stereotypes: JavaClass

This is a combination of Checkable and Enforceable Requirement.

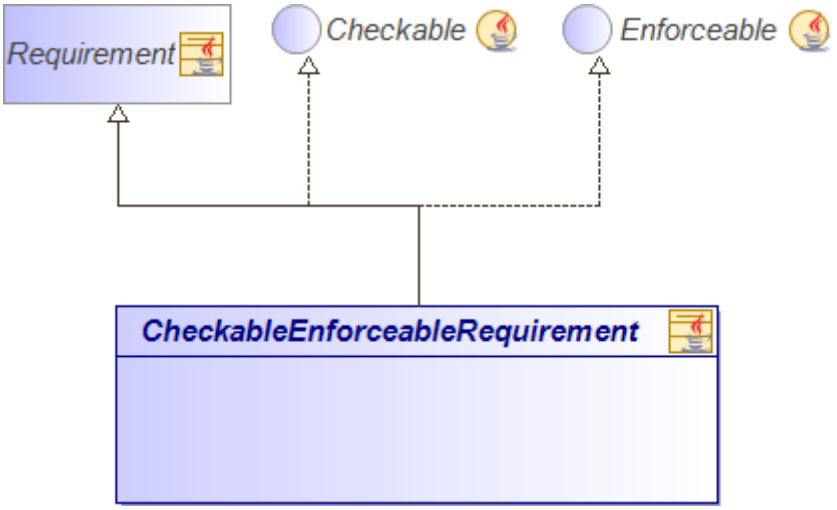

Figure 6 CheckableEnforceableRequirement (ClassStructureDiagramTemplate)

Package "rqcode.patterns.temporal"

*from Package rqcode.***patterns**

Stereotypes: JavaPackage

Contains implementation temporal patterns in RQCODE.

Name	Summary
GlobalUniversality	
Eventually	
GlobalResponseTimed	
GlobalResponseUntil	
GlobalUniversalityTimed	
AfterUntilUniversality	
MonitoringLoop	

Table 10 Owned Classes of Package "temporal"

Class "GlobalUniversality"

*from Package rqcode.patterns.***temporal**

Inherits from: MonitoringLoop

Stereotypes: JavaClass

Temporal requirements pattern:
Globally, Universally: Globally, it is always the case that P holds.

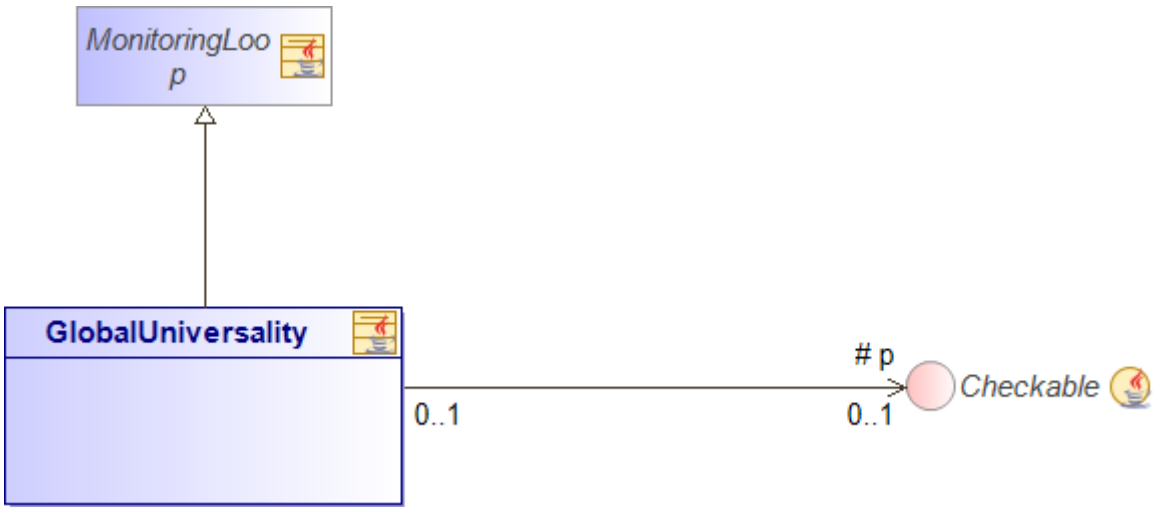

Figure 9 GlobalUniversality (ClassStructureDiagramTemplate)

Name	Description
GlobalUniversality (*inout* p Checkable)	
boolean invariant ()	
string toString ()	
string TCTL ()	

Table 11 Operations of Class "GlobalUniversality"

Name	Description
->p : [0..1] Checkable	

Table 12 Associations of Class "GlobalUniversality"

Class "Eventually"

*from Package rqcode.patterns.***temporal**

Inherits from: MonitoringLoop

Stereotypes: JavaClass

```
Temporal requirements pattern:
P always eventually holds
```

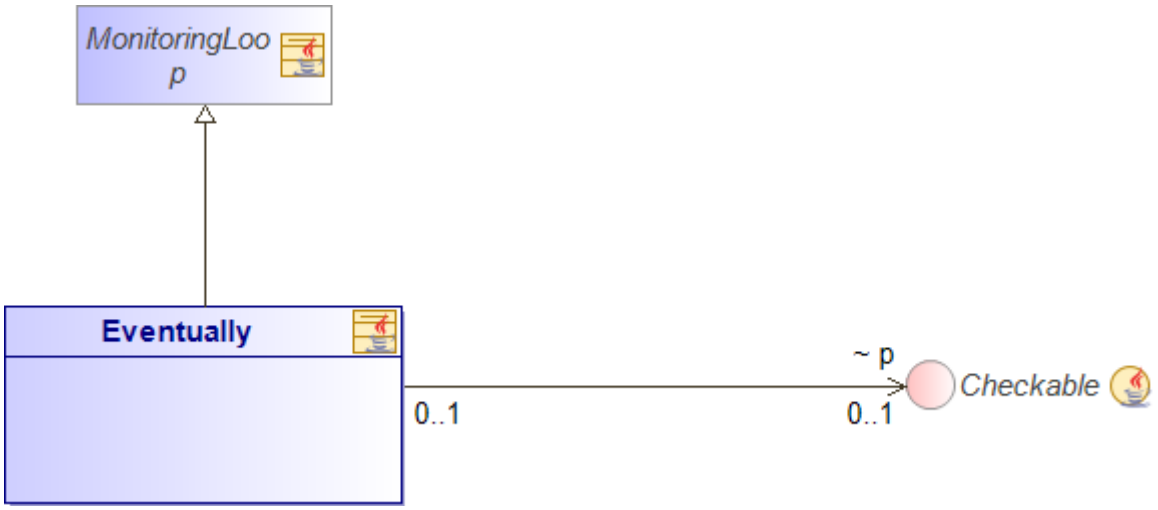

Figure 10 Eventually (ClassStructureDiagramTemplate)

Name	Description

Eventually (*Inout* p Checkable)
boolean exitCondition ()
boolean postcondition ()
string toString ()
string TCTL ()

Table 13 Operations of Class "Eventually"

Name	Description
->p : [0..1] Checkable	

Table 14 Associations of Class "Eventually"

Class "GlobalResponseTimed"

*from Package rqcode.patterns.***temporal**

Inherits from: MonitoringLoop

Stereotypes: JavaClass

Temporal requirements pattern:
Globally, Real-time Response: Globally, it is always the case that if P holds, the S eventually holds within T time units.

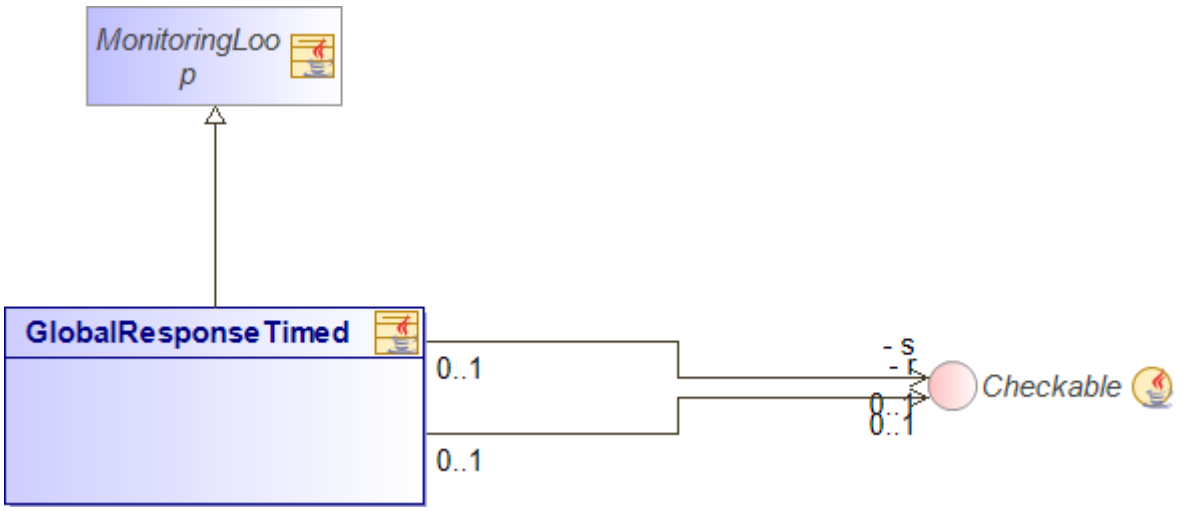

Figure 11 GlobalResponseTimed (ClassStructureDiagramTemplate)

Name	Description
GlobalResponseTimed (*Inout* s Checkable,*Inout* r Checkable,*Inout* boundary integer)	

boolean precondition ()
boolean postcondition ()
boolean exitCondition ()
string toString ()
string TCTL ()

Table 15 Operations of Class "GlobalResponseTimed"

Name	Description
->s : [0..1] Checkable	
->r : [0..1] Checkable	

Table 16 Associations of Class "GlobalResponseTimed"

Class "GlobalResponseUntil"

*from Package rqcode.patterns.***temporal**

Inherits from: MonitoringLoop

Stereotypes: JavaClass

Temporal requirements pattern:
Globally, it is always the case that if P holds then, unless R holds, Q will eventually hold

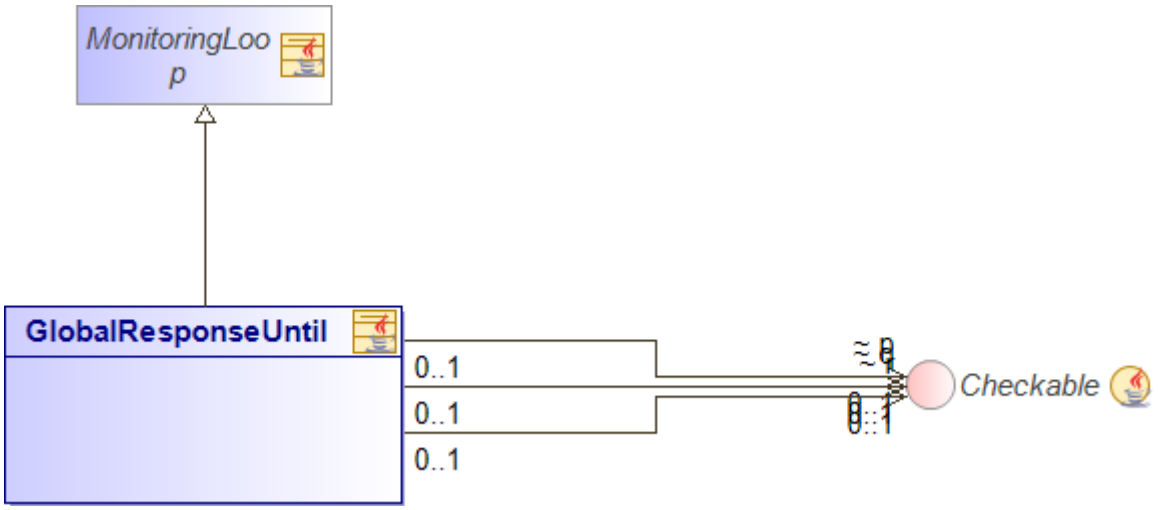

Figure 12 GlobalResponseUntil (ClassStructureDiagramTemplate)

Name	Description
GlobalResponseUntil (*Inout* p Checkable,*Inout* q Checkable,*Inout* r Checkable)	
boolean precondition ()	

boolean exitCondition ()
boolean postcondition ()
string toString ()
string TCTL ()

Table 17 Operations of Class "GlobalResponseUntil"

Name	Description
->p : [0..1] Checkable	
->q : [0..1] Checkable	
->r : [0..1] Checkable	

Table 18 Associations of Class "GlobalResponseUntil"

Class "GlobalUniversalityTimed"

*from Package rqcode.patterns.***temporal**

Inherits from: GlobalUniversality

Stereotypes: JavaClass

Temporal requirements pattern:
Timed Globally, Universally: Globally, it is always the case that if P held for T time units, then S holds.

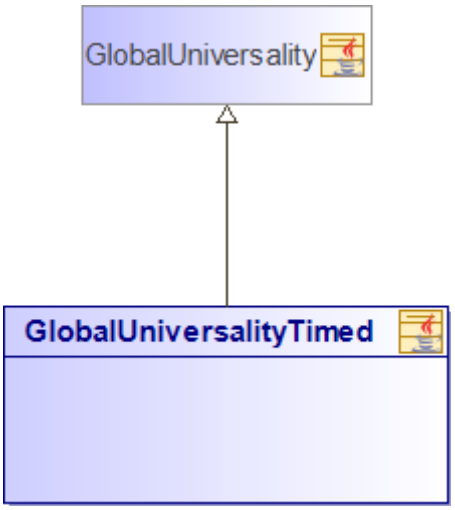

Figure 13 GlobalUniversalityTimed (ClassStructureDiagramTemplate)

Name	Description
GlobalUniversalityTimed (*Inout* p Checkable,*Inout* boundary integer)	
string toString ()	

string TCTL ()

Table 19 Operations of Class "GlobalUniversalityTimed"

Class "AfterUntilUniversality"

*from Package rqcode.patterns.***temporal**

Inherits from: MonitoringLoop

Stereotypes: JavaClass

Temporal requirements pattern:
After Q Until R Universally P: After Q, it is always the case that P holds until R holds.

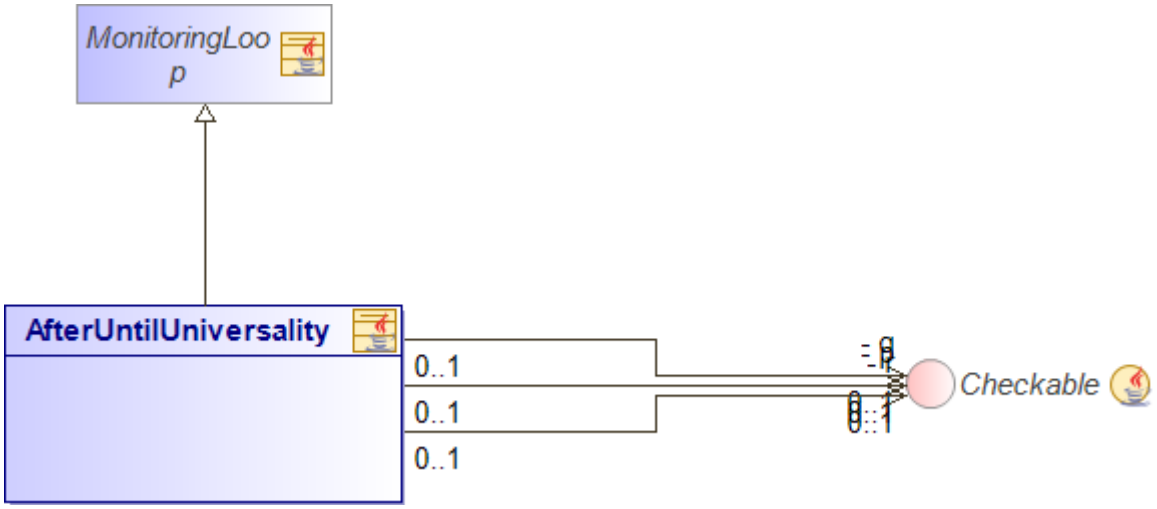

Figure 14 AfterUntilUniversality (ClassStructureDiagramTemplate)

Name	Description
AfterUntilUniversality (*Inout* q Checkable,*Inout* p Checkable,*Inout* r Checkable)	
boolean precondition ()	
boolean invariant ()	
boolean exitCondition ()	
string toString ()	
string TCTL ()	

Table 20 Operations of Class "AfterUntilUniversality"

Name	Description
->q : [0..1] Checkable	
->p : [0..1] Checkable	

->r : [0..1] Checkable

Table 21 Associations of Class "AfterUntilUniversality"

Class "MonitoringLoop"

*from Package rqcode.patterns.***temporal**

Implements: Checkable

Stereotypes: JavaClass

This is the monitoring service that periodically checks the temporal properties.

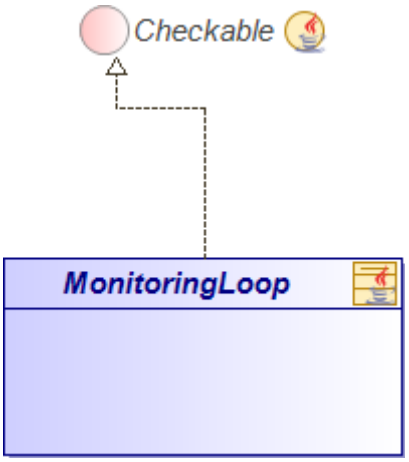

Figure 15 MonitoringLoop (ClassStructureDiagramTemplate)

Name	Description
integer variant (*Inout* i integer)	
integer sleepMilliseconds ()	
boolean invariant ()	
boolean precondition ()	
boolean postcondition ()	
boolean exitCondition ()	
CheckStatus check ()	
string TCTL ()	

Table 22 Operations of Class "MonitoringLoop"

Name	Description
boundary : [1..1] integer	

Table 23 Attributes of Class "MonitoringLoop"

Package "rqcode.patterns.win10"

*from Package rqcode.***patterns**

Stereotypes: JavaPackage

This are security requirements patterns that were created from STIGs.

Name	Summary
AccountManagementRequirement	
LogonRequirement	
UserAccountManagementRequirement	
LogonLogoffRequirement	
SensitivePrivilegeUseRequirement	
PrivilegeUseRequirement	
AuditPolicyRequirement	

Table 24 Owned Classes of Package "win10"

Class "AccountManagementRequirement"

*from Package rqcode.patterns.***win10**

Inherits from: AuditPolicyRequirement

Stereotypes: JavaClass

General STIG requirement pattern for checking settings of Win10 User Account Management functionality.

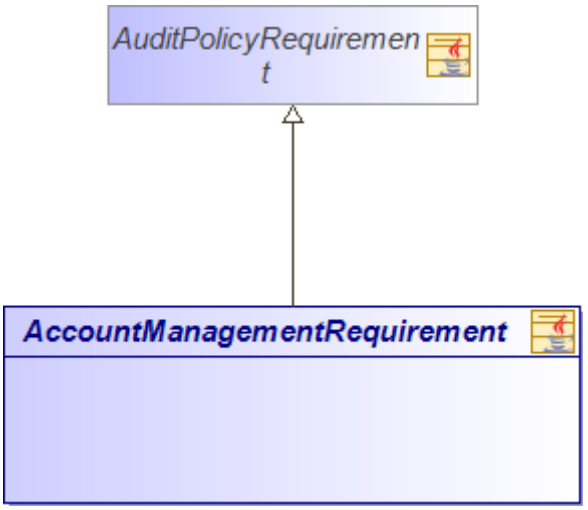

Figure 16 AccountManagementRequirement (ClassStructureDiagramTemplate)

Name	Description
string getCategory ()	

Table 25 Operations of Class "AccountManagementRequirement"

Class "LogonRequirement"

*from Package rqcode.patterns.***win10**

Inherits from: LogonLogoffRequirement

Stereotypes: JavaClass

General STIG requirement pattern for checking settings of Win10 Logon functionality.

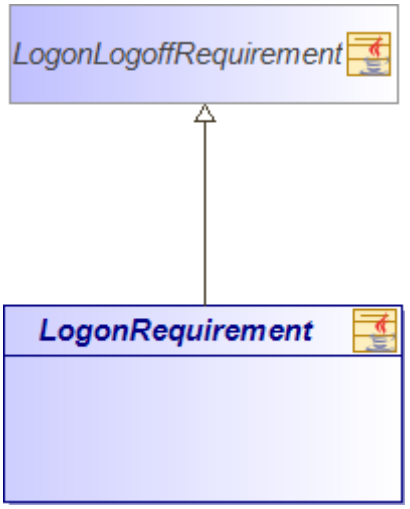

Figure 17 LogonRequirement (ClassStructureDiagramTemplate)

Name	Description
string getSubcategory ()	
string description ()	
string checkText ()	
string fixText ()	

Table 26 Operations of Class "LogonRequirement"

Class "UserAccountManagementRequirement"

*from Package rqcode.patterns.***win10**

Inherits from: AccountManagementRequirement

Stereotypes: JavaClass

General STIG requirement pattern for checking settings of Win10 User Account Management policies.

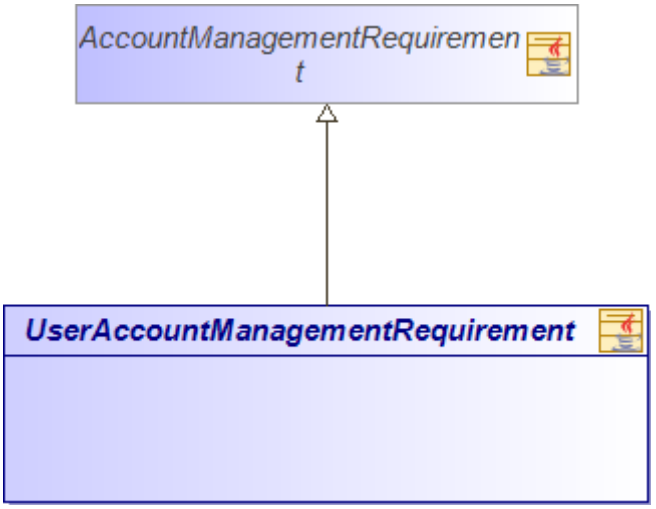

Figure 18 UserAccountManagementRequirement (ClassStructureDiagramTemplate)

Name	Description
string getSubcategory ()	
string description ()	
string checkText ()	
string fixText ()	

Table 27 Operations of Class "UserAccountManagementRequirement"

Class "LogonLogoffRequirement"

*from Package rqcode.patterns.***win10**

Inherits from: AuditPolicyRequirement

Stereotypes: JavaClass

General STIG requirement pattern for checking settings of Win10 Logon and Logoff functionality.

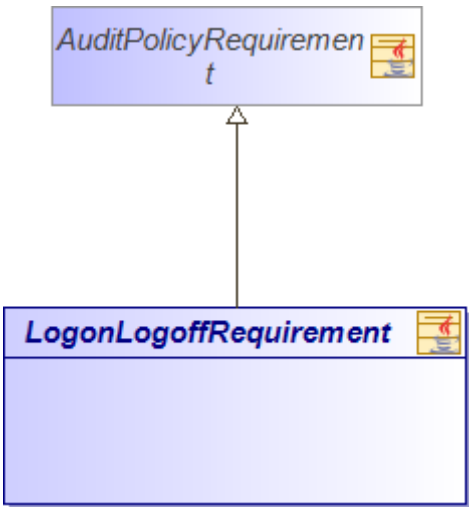

Figure 19 LogonLogoffRequirement (ClassStructureDiagramTemplate)

Name	Description
string getCategory ()	

Table 28 Operations of Class "LogonLogoffRequirement"

Class "SensitivePrivilegeUseRequirement"

*from Package rqcode.patterns.***win10**

Inherits from: PrivilegeUseRequirement

Stereotypes: JavaClass

General STIG requirement pattern for checking settings of Win10 Sensitive Privilege Use policies.

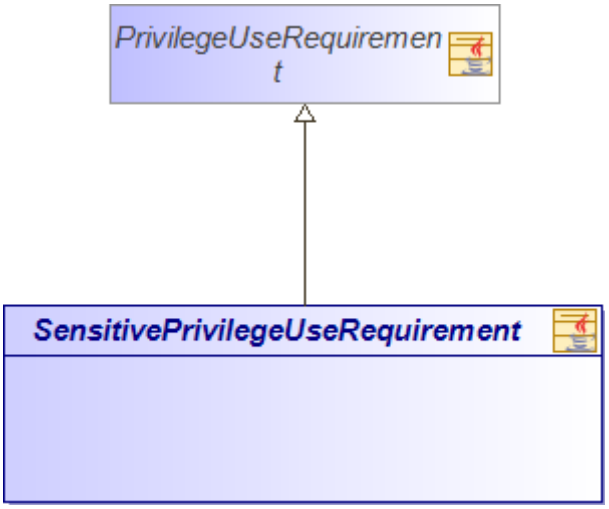

Figure 20 SensitivePrivilegeUseRequirement (ClassStructureDiagramTemplate)

Name	Description
string getSubcategory ()	
string description ()	
string checkText ()	
string fixText ()	

Table 29 Operations of Class "SensitivePrivilegeUseRequirement"

Class "PrivilegeUseRequirement"

***from Package rqcode.patterns.*win10**

Inherits from: AuditPolicyRequirement

Stereotypes: JavaClass

General STIG requirement pattern for checking settings of Win10 Privilege Use policies.

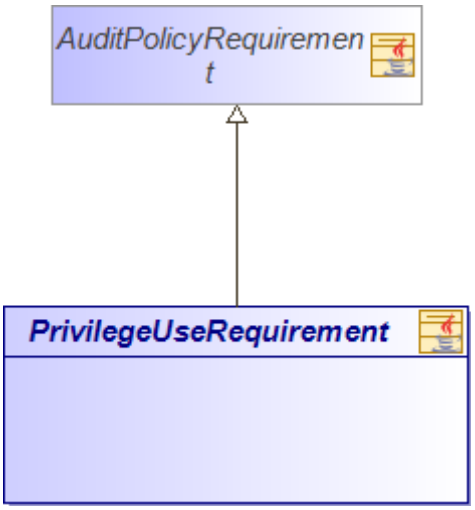

Figure 21 PrivilegeUseRequirement (ClassStructureDiagramTemplate)

Name	Description
string getCategory ()	

Table 30 Operations of Class "PrivilegeUseRequirement"

Class "AuditPolicyRequirement"

*from Package rqcode.patterns.***win10**

Inherits from: CheckableEnforceableRequirement

Stereotypes: JavaClass

Instances of this class are requirements related to Windows 10 audit policies. Instances of this utilize auditpol.exe to perform checking and enforcing; that is, they fork auditpol.exe manipulate its input and output. It would be ideal to perform checking and enforcing through Win32 API calls instead, but for the time being this approach works.

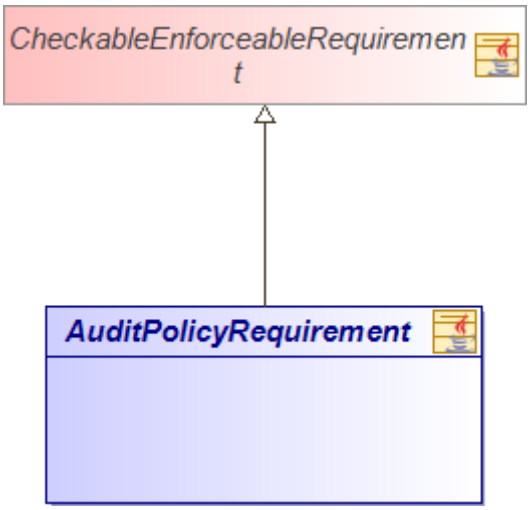

Figure 22 AuditPolicyRequirement (ClassStructureDiagramTemplate)

Name	Description
string getCategory ()	
string getSubcategory ()	
string getInclusionSetting ()	
string getSuccess ()	
string getFailure ()	
CheckStatus check ()	
EnforcementStatus enforce ()	

Table 31 Operations of Class "AuditPolicyRequirement"

Name	Summary
AuditPol	

Table 32 Owned Classes of Class "AuditPolicyRequirement"

Package "rqcode.stigs.ubuntu"

*from Package rqcode.***stigs**

Stereotypes: JavaPackage

Ubuntu related security requirements from STIG repository implemented in RQCODE.

Class "UbuntuPackagePattern"

*from Package rqcode.stigs.***ubuntu**

Implements: CheckableImplements: Enforceable

Stereotypes: JavaClass

RQCODE security requirements pattern from STIGS repository.

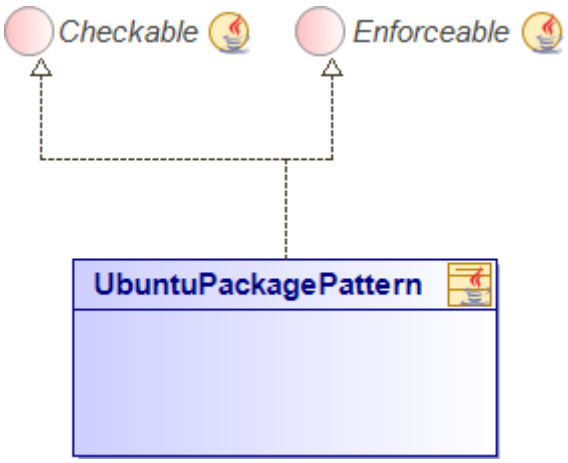

Figure 27 UbuntuPackagePattern (ClassStructureDiagramTemplate)

Name	Description
UbuntuPackagePattern (*Inout* name string,*Inout* mustBeInstalled boolean)	
CheckStatus check ()	
string toString ()	
EnforcementStatus enforce ()	

Table 36 Operations of Class "UbuntuPackagePattern"

Name	Description
_name : [1..1] string	
_mustBeInstalled : [1..1] boolean	

Table 37 Attributes of Class "UbuntuPackagePattern"

Class "Main"

*from Package rqcode.stigs.***ubuntu**

Stereotypes: JavaClass

Example of instantiation of the RQCODE for UBUNTU STIGs.

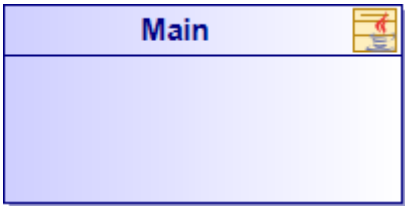

Figure 28 Main (ClassStructureDiagramTemplate)

Name	Description
main (*Inout* args string)	

Table 38 Operations of Class "Main"

Class "V_219157"

*from Package rqcode.stigs.ubuntu.***V_219157**

Implements: Checkable

Stereotypes: JavaClass

Removing the Network Information Service (NIS) package decreases the risk of the accidental (or intentional) activation of NIS or NIS+ services.

https://www.stigviewer.com/stig/canonical_ubuntu_18.04_lts/2021-06-16/finding/V-219157

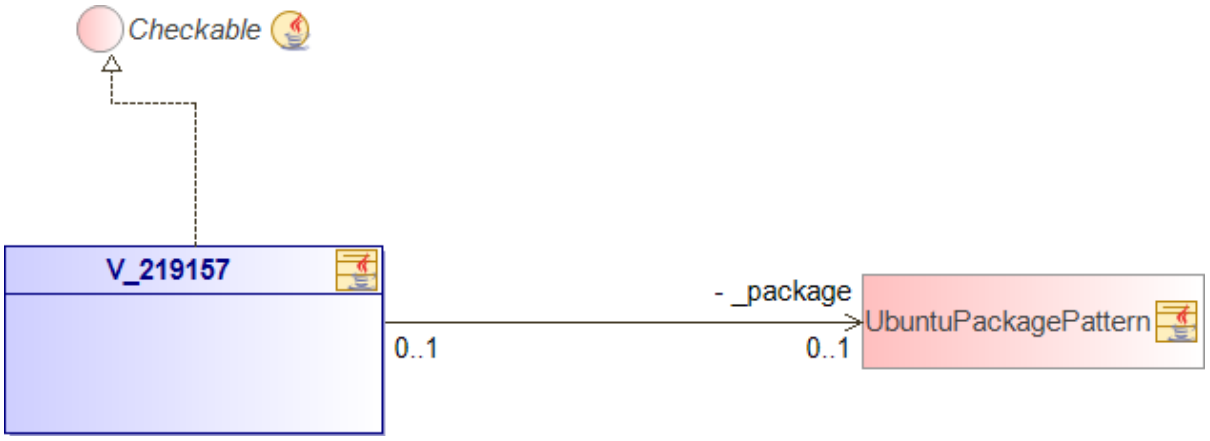

Figure 29 V_219157 (ClassStructureDiagramTemplate)

Name	Description
CheckStatus check ()	
string toString ()	

Table 40 Operations of Class "V_219157"

Name	Description
->_package : [0..1] UbuntuPackagePattern	

Table 41 Associations of Class "V_219157"

Class "V_219158"

*from Package rqcode.stigs.ubuntu.**V_219158***

Implements: Checkable

Stereotypes: JavaClass

It is detrimental for Ubuntu operating systems to provide, or install by default, functionality exceeding requirements or mission objectives. These unnecessary capabilities or services are often overlooked and therefore may remain unsecured. They increase the risk to the platform by providing additional attack vectors. Ubuntu operating systems are capable of providing a wide variety of functions and services. Some of the functions and services, provided by default, may not be necessary to support essential organizational operations (e.g., key missions, functions). The rsh-server service provides an unencrypted remote access service that does not provide for the confidentiality and integrity of user passwords or the remote session and has very weak authentication. If a privileged user were to log on using this service, the privileged user password could be compromised.

https://www.stigviewer.com/stig/canonical_ubuntu_18.04_lts/2021-06-16/finding/V-219158

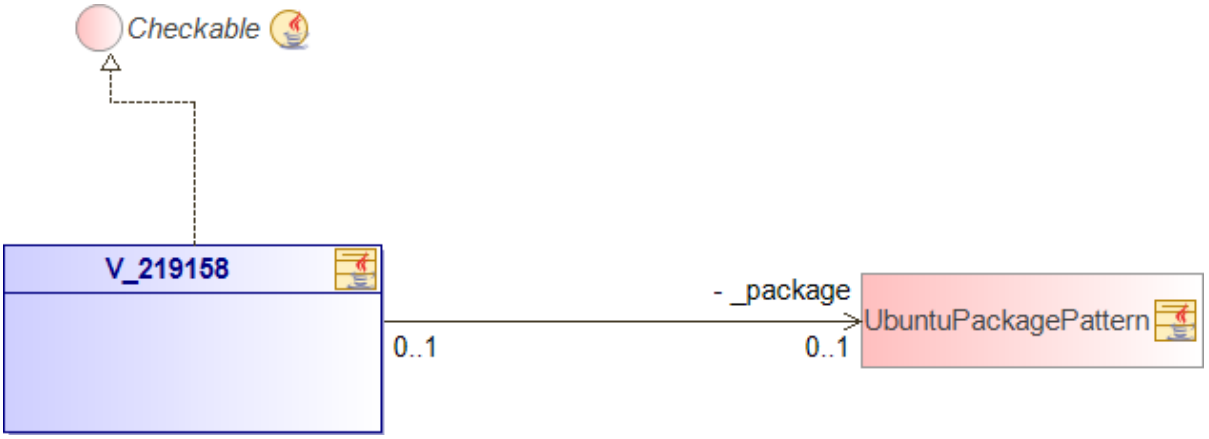

Figure 30 V_219158 (ClassStructureDiagramTemplate)

Name	Description
CheckStatus check ()	
string toString ()	

Table 43 Operations of Class "V_219158"

Name	Description
->_package : [0..1] UbuntuPackagePattern	

Table 44 Associations of Class "V_219158"

Class "V_219161"

*from Package rqcode.stigs.ubuntu.***V_219161**

Implements: Checkable

Stereotypes: JavaClass

Remote access services, such as those providing remote access to network devices and information systems, which lack automated control capabilities, increase risk and make remote user access management difficult at best. Remote access is access to DoD nonpublic information systems by an authorized user (or an information system) communicating through an external, non-organization-controlled network. Remote access methods include, for example, dial-up, broadband, and wireless. Ubuntu operating system functionality (e.g., RDP) must be capable of taking enforcement action if the audit reveals unauthorized activity. Automated control of remote

access sessions allows organizations to ensure ongoing compliance with remote access policies by enforcing connection rules of remote access applications on a variety of information system components (e.g., servers, workstations, notebook computers, smartphones, and tablets).

https://www.stigviewer.com/stig/canonical_ubuntu_18.04_lts/2021-06-16/finding/V-219161

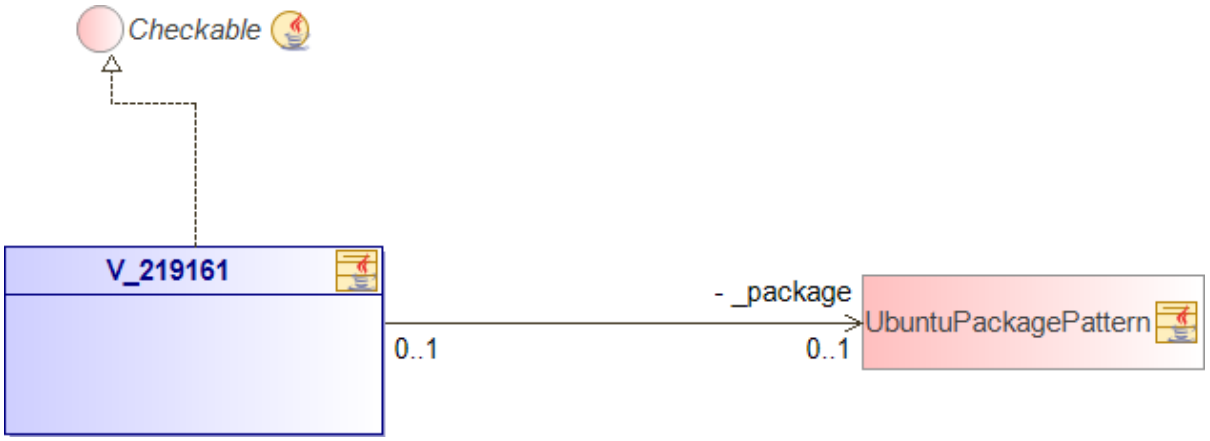

Figure 31 V_219161 (ClassStructureDiagramTemplate)

Name	Description
CheckStatus check ()	
string toString ()	

Table 46 Operations of Class "V_219161"

Name	Description
->_package : [0..1] UbuntuPackagePattern	

Table 47 Associations of Class "V_219161"

Class "V_219177"

*from Package rqcode.stigs.ubuntu.***V_219177**

Implements: Checkable

Stereotypes: JavaClass

Passwords need to be protected at all times, and encryption is the standard method for protecting passwords. If passwords are not encrypted, they can be plainly read (i.e., clear text) and easily compromised.

https://www.stigviewer.com/stig/canonical_ubuntu_18.04_lts/2021-06-16/finding/V-219177

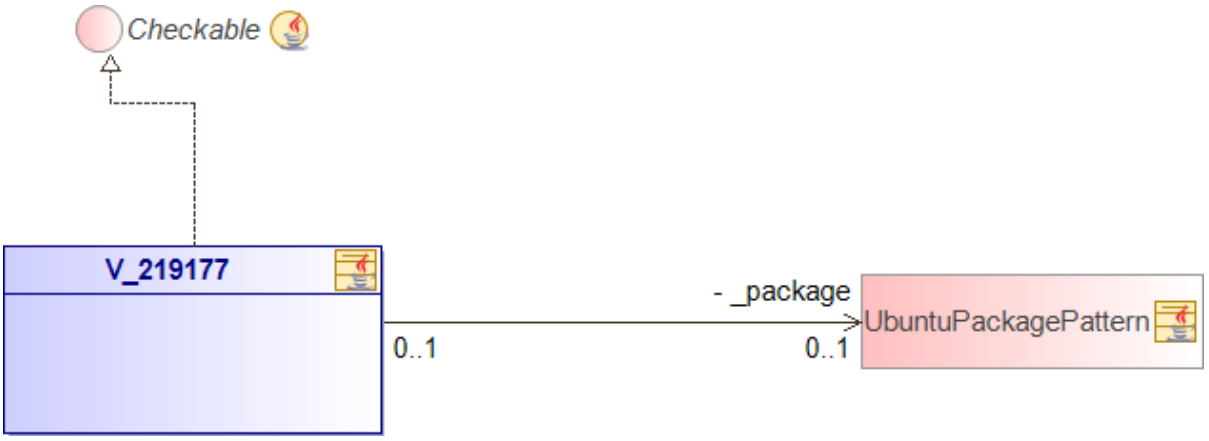

Figure 32 V_219177 (ClassStructureDiagramTemplate)

Name	Description
CheckStatus check ()	
string toString ()	

Table 49 Operations of Class "V_219177"

Name	Description
->_package : [0..1] UbuntuPackagePattern	

Table 50 Associations of Class "V_219177"

Class "V_219304"

*from Package rqcode.stigs.ubuntu.***V_219304**

Implements: Checkable

Stereotypes: JavaClass

A session lock is a temporary action taken when a user stops work and moves away from the immediate physical vicinity of the information system but does not want to log out because of the temporary nature of the absence. The session lock is implemented at the point where session activity can be determined. Rather than be forced to wait for a period of time to expire before the user session can be locked, the Ubuntu operating system need to provide users with the ability to manually invoke a session lock so users may secure their session should the need arise for them to temporarily vacate the immediate physical vicinity. Satisfies: SRG-OS-000030-GPOS-00011, SRG-OS-000031-GPOS-00012.

https://www.stigviewer.com/stig/canonical_ubuntu_18.04_lts/2021-06-16/finding/V-219304

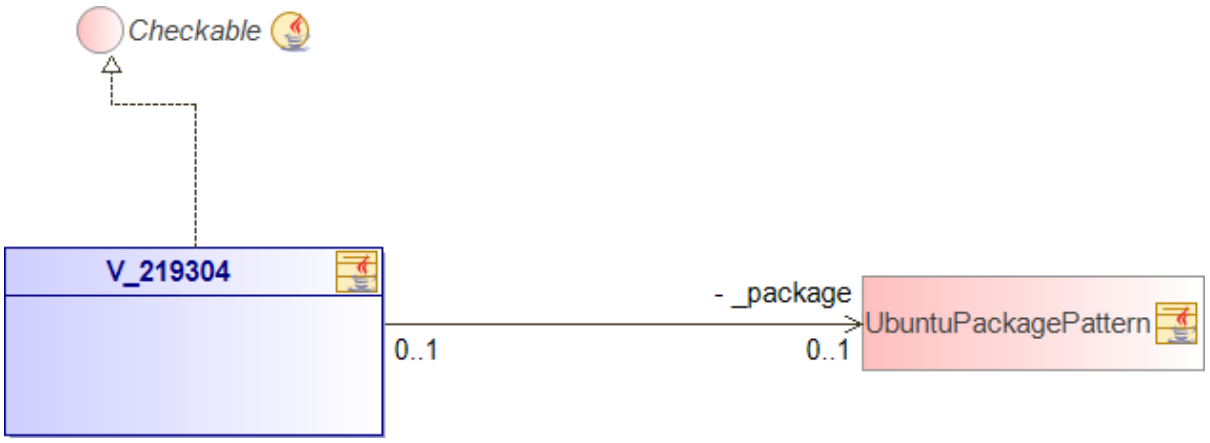

Figure 33 V_219304 (ClassStructureDiagramTemplate)

Name	Description
CheckStatus check ()	
string toString ()	

Table 52 Operations of Class "V_219304"

Name	Description
->_package : [0..1] UbuntuPackagePattern	

Table 53 Associations of Class "V_219304"

Class "V_219318"

*from Package rqcode.stigs.ubuntu.***V_219318**

Implements: Checkable

Stereotypes: JavaClass

Using an authentication device, such as a CAC or token that is separate from the information system, ensures that even if the information system is compromised, that compromise will not affect credentials stored on the authentication device. Multifactor solutions that require devices separate from information systems gaining access include, for example, hardware tokens providing time-based or challenge-response authenticators and smart cards such as the U.S. Government Personal Identity Verification card and the DoD Common Access Card. A privileged account is defined as an information system account with authorizations of a privileged user. Remote access is access to DoD nonpublic information systems by an authorized user (or an information system) communicating

through an external, non-organization-controlled network. Remote access methods include, for example, dial-up, broadband, and wireless. This requirement only applies to components where this is specific to the function of the device or has the concept of an organizational user (e.g., VPN, proxy capability). This does not apply to authentication for the purpose of configuring the device itself (management). Requires further clarification from NIST.

https://www.stigviewer.com/stig/canonical_ubuntu_18.04_lts/2021-06-16/finding/V-219318

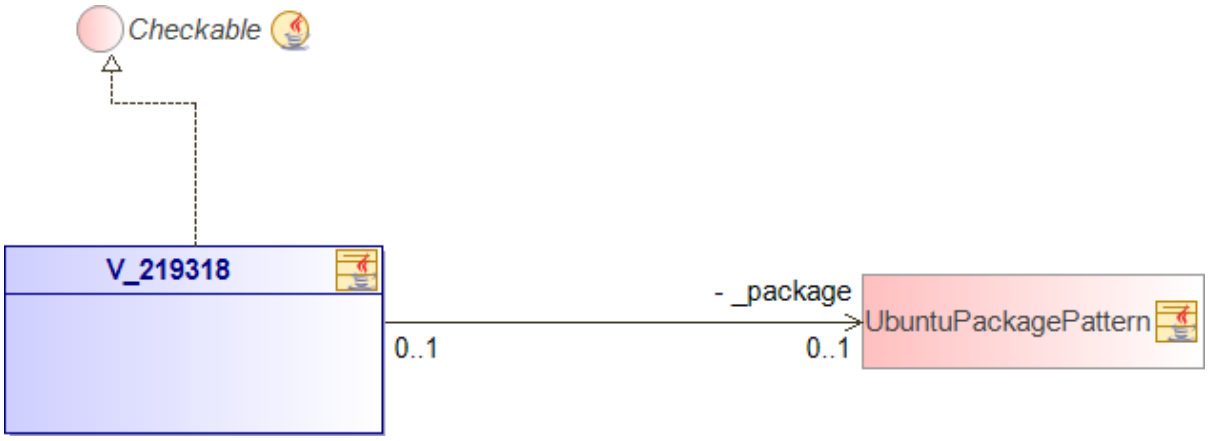

Figure 34 V_219318 (ClassStructureDiagramTemplate)

Name	Description
CheckStatus check ()	
string toString ()	

Table 55 Operations of Class "V_219318"

Name	Description
->_package : [0..1] UbuntuPackagePattern	

Table 56 Associations of Class "V_219318"

Class "V_219319"

*from Package rqcode.stigs.ubuntu.**V_219319***

Implements: Checkable

Stereotypes: JavaClass

The use of PIV credentials facilitates standardization and reduces the risk of unauthorized access. DoD has mandated the use of the CAC to support identity management and personal authentication for systems covered under Homeland Security Presidential Directive (HSPD) 12, as well as making the CAC a primary component of layered protection for national security systems.

https://www.stigviewer.com/stig/canonical_ubuntu_18.04_lts/2021-06-16/finding/V-219319

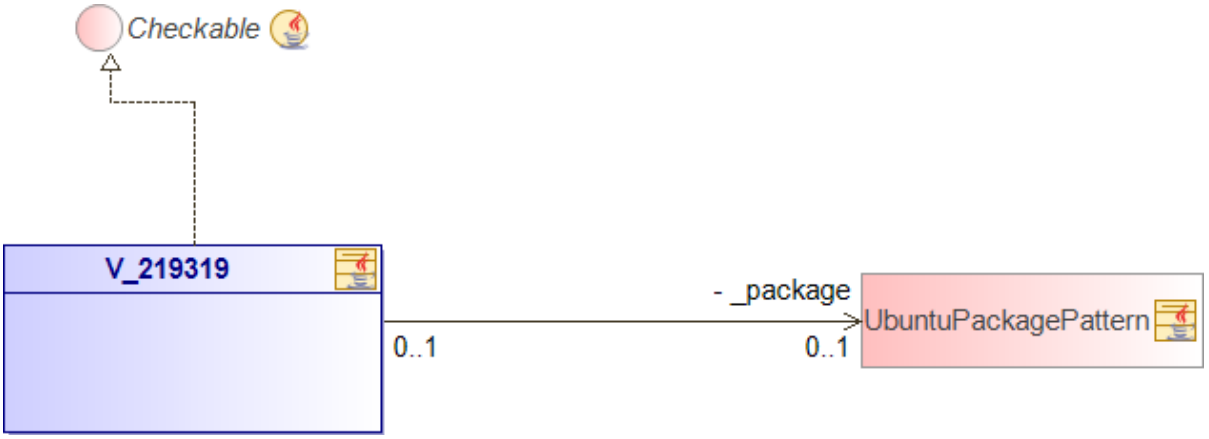

Figure 35 V_219319 (ClassStructureDiagramTemplate)

Name	Description
CheckStatus check ()	
string toString ()	

Table 58 Operations of Class "V_219319"

Name	Description
->_package : [0..1] UbuntuPackagePattern	

Table 59 Associations of Class "V_219319"

Class "V_219343"

*from Package rqcode.stigs.ubuntu.***V_219343**

Implements: Checkable

Stereotypes: JavaClass

Without verification of the security functions, security functions may not operate correctly and the failure may go unnoticed. Security function is defined as the hardware, software, and/or firmware of

the information system responsible for enforcing the system security policy and supporting the isolation of code and data on which the protection is based. Security functionality includes, but is not limited to, establishing system accounts, configuring access authorizations (i.e., permissions, privileges), setting events to be audited, and setting intrusion detection parameters. This requirement applies to the Ubuntu operating system performing security function verification/testing and/or systems and environments that require this functionality.

https://www.stigviewer.com/stig/canonical_ubuntu_18.04_lts/2021-06-16/finding/V-219343

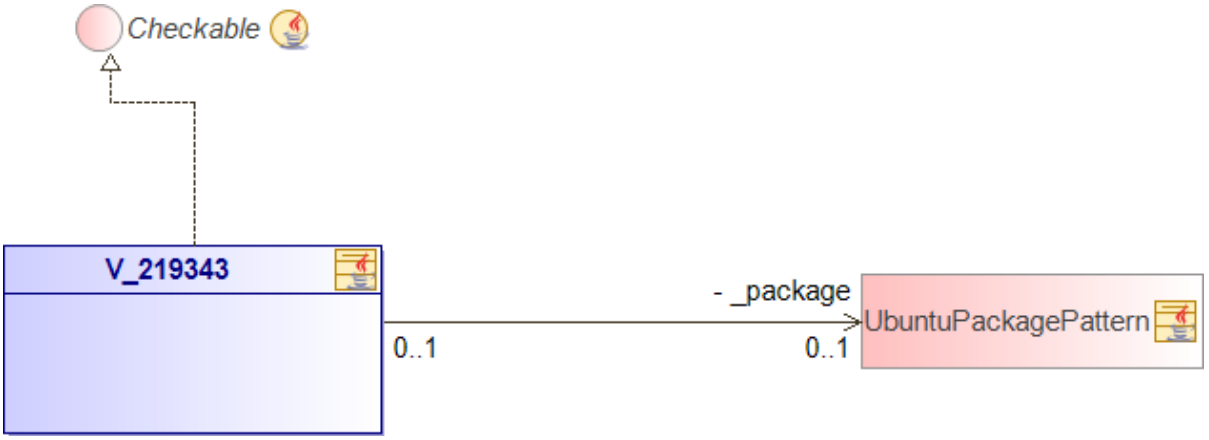

Figure 36 V_219343 (ClassStructureDiagramTemplate)

Name	Description
CheckStatus check ()	
string toString ()	

Table 61 Operations of Class "V_219343"

Name	Description
->_package : [0..1] UbuntuPackagePattern	

Table 62 Associations of Class "V_219343"

Package "rqcode.stigs.win10"

*from Package rqcode.**stigs***

Class "V_63487"

*from Package rqcode.stigs.**win10***

Inherits from: SensitivePrivilegeUseRequirement

Stereotypes: JavaClass

Maintaining an audit trail of system activity logs can help identify configuration errors, troubleshoot service disruptions, and analyze compromises that have occurred, as well as detect attacks. Audit logs are necessary to provide a trail of evidence in case the system or network is compromised. Collecting this data is essential for analyzing the security of information assets and detecting signs of suspicious and unexpected behavior. Sensitive Privilege Use records events related to use of sensitive privileges, such as "Act as part of the operating system" or "Debug programs".

https://www.stigviewer.com/stig/windows_10/2016-10-28/finding/V-63487

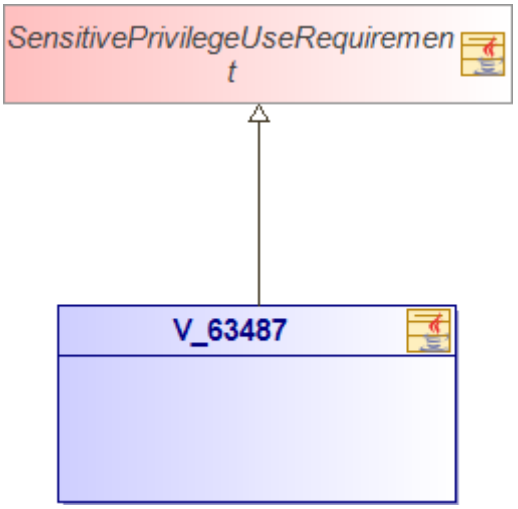

Figure 37 V_63487 (ClassStructureDiagramTemplate)

Name	Description
string getFailure ()	
string getInclusionSetting ()	
string getSuccess ()	
string checkTextCode ()	
string date ()	
string findingID ()	

string fixTextCode ()
string iAControls ()
string ruleID ()
string sTIG ()
string severity ()
string version ()

Table 64 Operations of Class "V_63487"

Class "Windows10SecurityTechnicalImplementationGuide"

*from Package rqcode.stigs.***win10**

Stereotypes: JavaClass

This is example of the instantiation of the Win 10 STIG requirements.

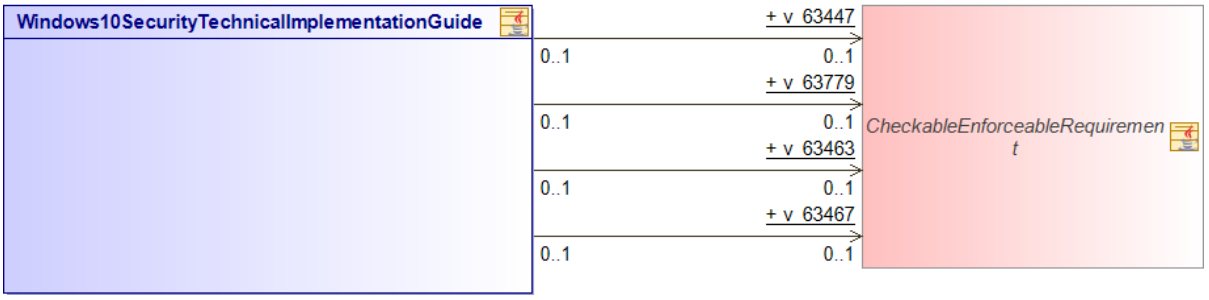

Figure 38 Windows10SecurityTechnicalImplementationGuide (ClassStructureDiagramTemplate)

Name	Description
CheckableEnforceableRequirement allSTIGs ()	

Table 65 Operations of Class "Windows10SecurityTechnicalImplementationGuide"

Name	Description
->v_63447 : [0..1] CheckableEnforceableRequirement	
->v_63779 : [0..1] CheckableEnforceableRequirement	
->v_63463 : [0..1] CheckableEnforceableRequirement	
->v_63467 : [0..1] CheckableEnforceableRequirement	

Table 66 Associations of Class "Windows10SecurityTechnicalImplementationGuide"

Class "V_63449"

*from Package rqcode.stigs.***win10**

Inherits from: UserAccountManagementRequirement

Stereotypes: JavaClass

Maintaining an audit trail of system activity logs can help identify configuration errors, troubleshoot service disruptions, and analyze compromises that have occurred, as well as detect attacks. Audit logs are necessary to provide a trail of evidence in case the system or network is compromised. Collecting this data is essential for analyzing the security of information assets and detecting signs of suspicious and unexpected behavior. User Account Management records events such as creating, changing, deleting, renaming, disabling, or enabling user accounts.

https://www.stigviewer.com/stig/windows_10/2016-10-28/finding/V-63449

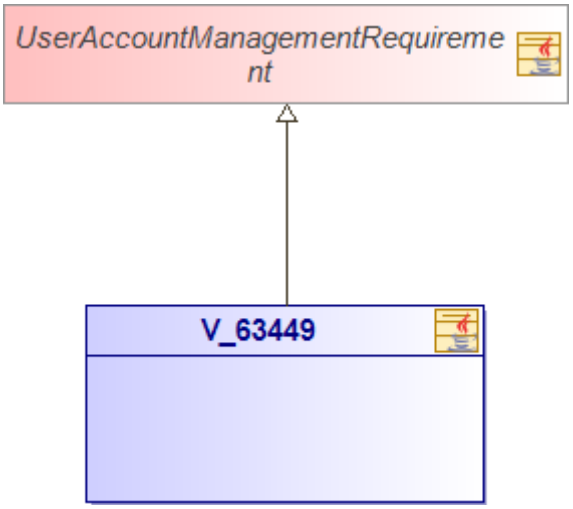

Figure 39 V_63449 (ClassStructureDiagramTemplate)

Name	Description
string findingID ()	
string version ()	
string ruleID ()	
string iAControls ()	
string severity ()	
string sTIG ()	
string date ()	
string checkTextCode ()	
string fixTextCode ()	
string getInclusionSetting ()	

string getSuccess ()
string getFailure ()

Table 67 Operations of Class "V_63449"

Class "V_63463"

*from Package rqcode.stigs.***win10**

Inherits from: LogonRequirement

Stereotypes: JavaClass

Maintaining an audit trail of system activity logs can help identify configuration errors, troubleshoot service disruptions, and analyze compromises that have occurred, as well as detect attacks. Audit logs are necessary to provide a trail of evidence in case the system or network is compromised. Collecting this data is essential for analyzing the security of information assets and detecting signs of suspicious and unexpected behavior. Logon records user logons. If this is an interactive logon, it is recorded on the local system. If it is to a network share, it is recorded on the system accessed.

https://www.stigviewer.com/stig/windows_10/2016-10-28/finding/V-63463

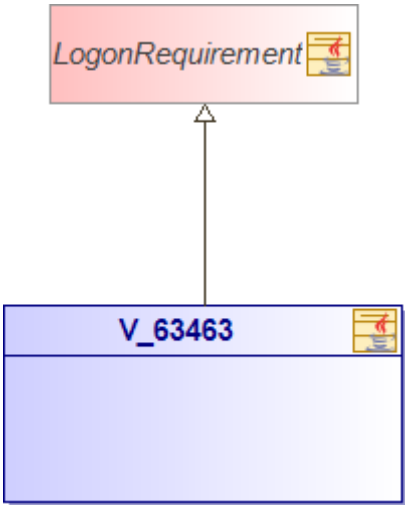

Figure 40 V_63463 (ClassStructureDiagramTemplate)

Name	Description
string getFailure ()	
string getInclusionSetting ()	
string getSuccess ()	
string checkTextCode ()	
string date ()	

string findingID ()
string fixTextCode ()
string iAControls ()
string ruleID ()
string sTIG ()
string severity ()
string version ()

Table 68 Operations of Class "V_63463"

Class "V_63483"

*from Package rqcode.stigs.***win10**

Inherits from: SensitivePrivilegeUseRequirement

Stereotypes: JavaClass

Maintaining an audit trail of system activity logs can help identify configuration errors, troubleshoot service disruptions, and analyze compromises that have occurred, as well as detect attacks. Audit logs are necessary to provide a trail of evidence in case the system or network is compromised. Collecting this data is essential for analyzing the security of information assets and detecting signs of suspicious and unexpected behavior. Sensitive Privilege Use records events related to use of sensitive privileges, such as "Act as part of the operating system" or "Debug programs".

https://www.stigviewer.com/stig/windows_10/2016-10-28/finding/V-63483

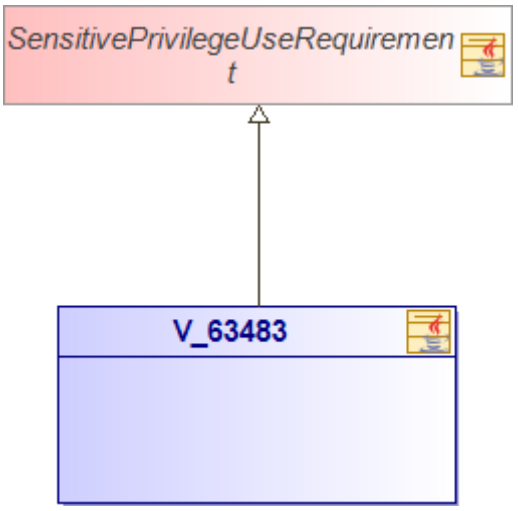

Figure 41 V_63483 (ClassStructureDiagramTemplate)

Name	Description

string getFailure ()
string getInclusionSetting ()
string getSuccess ()
string checkTextCode ()
string date ()
string findingID ()
string fixTextCode ()
string iAControls ()
string ruleID ()
string sTIG ()
string severity ()
string version ()

Table 69 Operations of Class "V_63483"

Class "V_63467"

*from Package rqcode.stigs.***win10**

Inherits from: LogonRequirement

Stereotypes: JavaClass

Maintaining an audit trail of system activity logs can help identify configuration errors, troubleshoot service disruptions, and analyze compromises that have occurred, as well as detect attacks. Audit logs are necessary to provide a trail of evidence in case the system or network is compromised. Collecting this data is essential for analyzing the security of information assets and detecting signs of suspicious and unexpected behavior. Logon records user logons. If this is an interactive logon, it is recorded on the local system. If it is to a network share, it is recorded on the system accessed.

https://www.stigviewer.com/stig/windows_10/2016-10-28/finding/V-63467

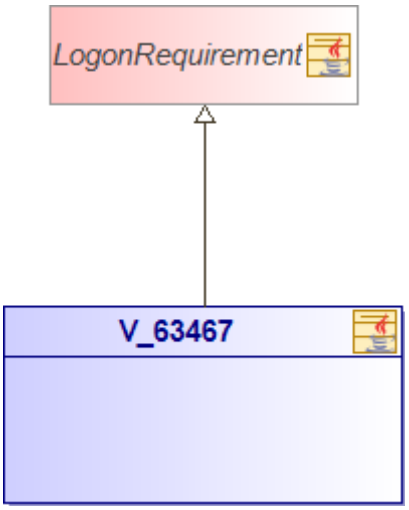

Figure 42 V_63467 (ClassStructureDiagramTemplate)

Name	Description
string getFailure ()	
string getInclusionSetting ()	
string getSuccess ()	
string checkTextCode ()	
string date ()	
string findingID ()	
string fixTextCode ()	
string iAControls ()	
string ruleID ()	
string sTIG ()	
string severity ()	
string version ()	

Table 70 Operations of Class "V_63467"

Class "V_63447"

*from Package rqcode.stigs.***win10**

Inherits from: UserAccountManagementRequirement

Stereotypes: JavaClass

Maintaining an audit trail of system activity logs can help identify configuration errors, troubleshoot service disruptions, and analyze compromises that have occurred, as well as detect attacks. Audit logs are necessary to provide a trail of evidence in case the system or network is compromised. Collecting this data is essential for analyzing the security of information assets and detecting signs of suspicious and unexpected behavior. User Account Management records events such as creating,

changing, deleting, renaming, disabling, or enabling user accounts.

https://www.stigviewer.com/stig/windows_10/2016-10-28/finding/V-63447

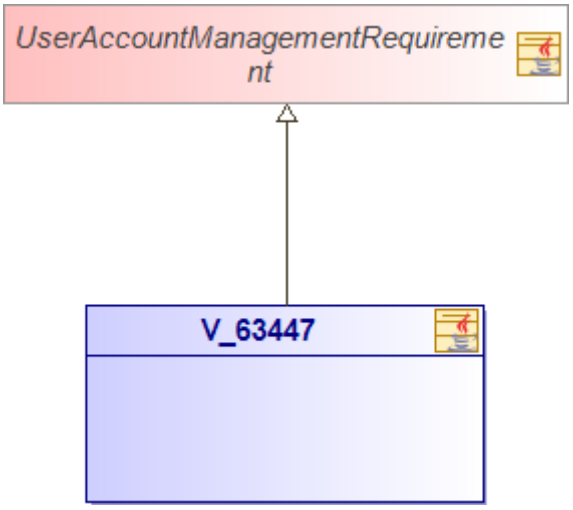

Figure 43 V_63447 (ClassStructureDiagramTemplate)

Name	Description
string findingID ()	
string version ()	
string ruleID ()	
string iAControls ()	
string severity ()	
string sTIG ()	
string date ()	
string checkTextCode ()	
string fixTextCode ()	
string getInclusionSetting ()	
string getSuccess ()	
string getFailure ()	

Table 71 Operations of Class "V_63447"