National Test Institute for Cybersecurity

Test Report

Security Audit QGIS Server and QWC2

Version	1.0
Date	13.10.2025
Customer	National Cyber Security Centre NCSC
Classification	Public

Table of Contents

Version	Date	Description	Author
1.0	13.10.2025	Publication	Brian Ceccato
0.4	30.09.2025	Update regarding fix for H2 and CVE IDs	Fabio Zuber
0.3	12.09.2025	Draft for publication	Fabio Zuber
0.2	26.06.2025	Draft of full report (for review by NCSC)	Fabio Zuber
0.1	03.06.2025	Preliminary report for disclosure to QGIS security team	Fabio Zuber

National Test Institute for Cybersecurity NTC Baarerstrasse 53 6300 Zug +41 41 317 00 11 office@ntc.swiss www.ntc.swiss	**Fabio Zuber** Penetration Tester +41 41 317 00 14 fabio.zuber@ntc.swiss

1 Management Summary

1.1 Introduction and Context

This report presents the results of the security assessment of QGIS Server and QWC2 carried out by the Swiss National Test Institute for Cybersecurity NTC on behalf of the Swiss National Cyber Security Centre (NCSC).

The NCSC selected the QGIS project for this security assessment based on the results of a survey conducted from January 14 to February 28, 2025. The survey was published on the Cyber Security Hub and invited federal, cantonal, and municipal authorities to propose open source software for testing. QGIS was chosen due to its widespread use across all three administrative levels and its role in processing potentially sensitive data.

QGIS is a powerful open source Geographic Information System (GIS) widely adopted by the Swiss public administration. Government organizations use it to visualize and analyze spatial data for essential tasks like urban planning, managing public infrastructure, and conducting environmental assessments.

The QGIS Server acts as a backend for collaborating with others on GIS data. QWC2 (QGIS Web Client), on the other hand, is a web application for editing map data. There is also a desktop client for QGIS that is commonly used by public administrations. However, this analysis focused on the server and web front end because they are more at risk since they are typically exposed directly to the internet.

The NTC performed a security analysis of QGIS Server and QWC2 between April 2025 and Mai 2025. As part of the tests, the source code was reviewed for potential security issues. This analysis was conducted using manual and automated testing procedures. The main goals of the analysis were to ensure that sensitive data cannot be accessed by unauthorized parties and that data manipulation is not possible without proper authorization.

The QGIS development community was informed in advance by the NCSC about the upcoming analysis. During the testing period, the QGIS maintainers, the NTC and the NCSC worked together and shared information, giving the NTC valuable technical insights.
All risks identified in the analysis were reported by the NCSC to the QGIS security team and addressed subsequently. By the time of publication, all reported vulnerabilities were fixed and releases to fix the vulnerabilities were made available.

1.2 Assessment Summary

The penetration test confirmed the strong security posture of the QGIS server. Within the testing timeframe, no directly exploitable vulnerabilities were identified.

The only identified security issue has a low criticality and was found by analyzing the source code. QGIS's dependencies do not explicitly specify which version is expected to be installed. This may lead to unpredictable build failures and security issues in the event of a supply chain attack.

Five potential security issues were identified in the Web UI called "QWC2". Two of these vulnerabilities, both categorized as "High", allow attackers to insert arbitrary JavaScript code, which would then be executed when another user accesses the affected part of the application. Such cross-site scripting (XSS) vulnerabilities are typically exploited to steal user sessions, especially those with elevated permissions. However, since attackers would require a valid account and editing permissions to exploit these vulnerabilities, the likelihood of a real-world attack is reduced.

The distribution of findings between QGIS Server and QWC2 is illustrated in Figure 1.

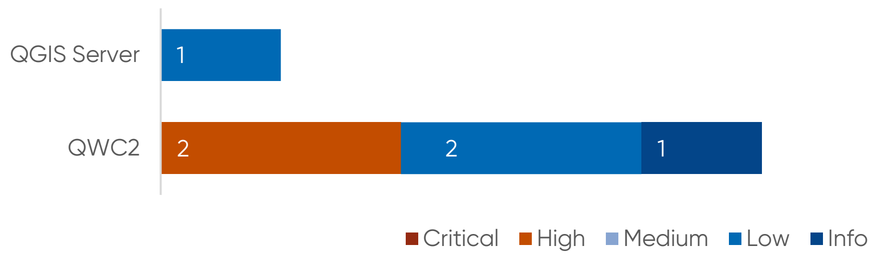


Figure 1: Distribution of security issues by component

In addition to the aforementioned XSS vulnerabilities, suboptimal cookie security attributes were observed in the QWC2 admin UI. While this does not constitute a vulnerability in itself, a more secure default configuration is recommended.

All details about the identified risks and recommendations can be found in *Chapter 2.4.*

The QGIS team promptly addressed the reported risks and issued fixes where required. Users should ensure they are running the latest stable versions of QGIS and QWC2.

We want to thank the NCSC and the QGIS security team for their professional handling of these issues and their continued efforts in strengthening the security of the open source ecosystem.

2 Scope, Findings and Recommendations

2.1 Overview of Tested Components

The NTC conducted a security assessment of QGIS to identify potential vulnerabilities. The testing scope included both the QGIS Server and the QWC2 web application. QWC2 is one of many frontend applications used to visualize and interact with GIS data. It was included in the scope because it is used by multiple cantons, is fully open source and is provided by the same community that develops QGIS Server.

The solution was tested from April 1 to May 30, 2025, using both manual and automated methods. The total effort was approximately 20 person-days, conducted by two testing specialists.

The dynamic testing was primarily performed manually by experienced security experts using tools such as the Burp Suite Proxy, various browser extensions, and a variety of specialized tools. Where applicable and appropriate, testing followed the OWASP Application Security Verification Standard (ASVS). In addition, automated analysis tools such as the Burp Suite Active Scanner were used where appropriate to efficiently identify known vulnerabilities and misconfigurations.

All dynamic tests were conducted using a local installation of QGIS Server and QWC2. The NTC followed the official quick start guide to run the services locally using Docker. All tests were conducted using the default settings and no modifications to security-related configurations.

The static source code analysis was performed on the security-relevant code sections. This analysis focused on identifying vulnerabilities, rather than enforcing coding best practices, verifying documentation completeness or optimizing code efficiency. Testing was performed using a combination of automated and manual checks. Automated testing was carried out using scanners such as SonarQube, which efficiently identify known vulnerabilities and coding errors.

Since QGIS Server and the QGIS desktop client are developed in a common git repository[1], the source code analysis was performed on that code base, focusing on the shared logic and the server-specific components. The code that is used only for the desktop client was not part of this analysis.

[1] https://github.com/qgis/QGIS

2.2 People Involved

Rolle	Persons
Project lead NCSC	Roger Knoepfel
Project Lead NTC	Fabio Zuber
Testing	Patrik Fabian, Fabio Zuber
Quality Assurance	Tobias Castagna

2.3 Project Timeline

Task	Date	Persons involved
Kick-off Meeting OSS Pilot Project	14.11.2025	NCSC Vulnerability Management Team **NTC**: Tobias Castagna, Dilip Many, Fabio Zuber
Testing and code analysis	01.04.2025 – 30.05.2025	Patrik Fabian, Fabio Zuber
Handover preliminary vulnerability report to QGIS security team	05.06.2025	NCSC Vulnerability Management Team
Correction / Clarification for QWC2 Registration finding	31.07.2025	Fabio Zuber
Final report	12.09.2025	Tobias Castagna, Andreas Leisibach, Dilip Many, Fabio Zuber
Last fixes implemented and release made available	30.09.2025	QGIS Maintainers
CVE IDs reserved for CVE-2025-11183 and CVE-2025-11184	30.09.2025	NCSC Vulnerability Management Team
Publication of report and CVEs	13.10.2025	NCSC Vulnerability Management Team

2.4 Penetration Test Details

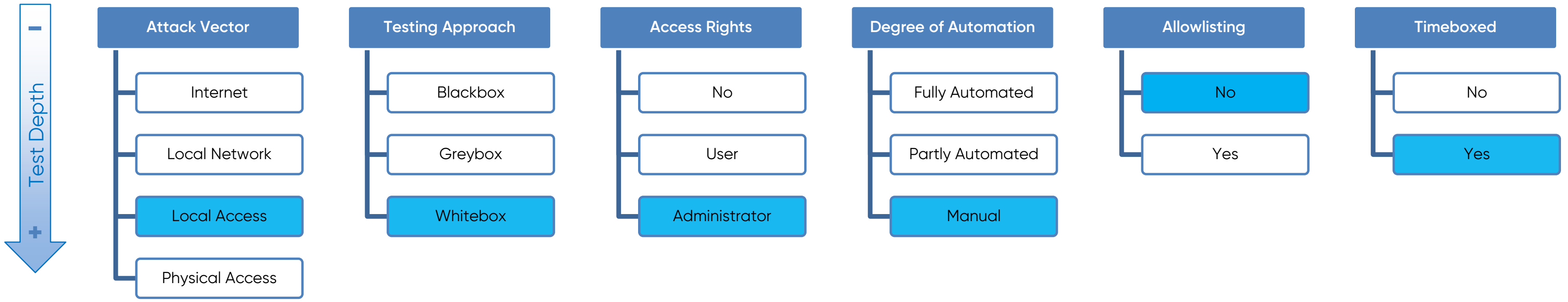
2.4.1 Scope and Tested Version

The tests were carried out between April 1 to May 30, 2025. The table below lists the tested components and their tested version. The code analysis and dynamic security testing focused on code that is used to run QGIS Server and QWC2. Code used for quality assurance, testing and automation was excluded from the analysis.

Project Name	Source Repository	Tested Version	Remarks
QGIS	https://github.com/qgis/QGIS	final-3_42_3 (e84bda9)	Used for source code review This analysis focused on the server components. The desktop client was not part of this analysis.
QWC2	https://github.com/qgis/qwc2	v2025.11-lts (df80336)	Used for Source Code Review
qwc-docker	https://github.com/qwc-services/qwc-docker	master (c6203bf)	Docker compose setup used for dynamic testing. Following the quick start guide from: https://qwc-services.github.io/master/QuickStart/

2.4.2 Test Conditions

The security audit took place under these conditions:



				No additional WAF or firewall was used during the tests. No security settings were disabled in the app settings.	Testing was done in ~20 person days and done by two test experts. The available time was invested in areas deemed most impactful.

Further information about the graphic and the interpretation of the given values can be found in Appendix *3.2, Definitions for Test Conditions*.

The following tools were used to test QGIS Server and QWC2.

Name	Version	Reference	Remarks / Usage
BurpSuite Professional	2025.2.3	*https://portswigger.net/burp/pro*	Dynamic Testing: Web app and API
SonarQube Developer Edition	10.8	*https://www.sonarsource.com/products/sonarqube/*	Static code analysis
Snyk.io	-	*https://snyk.io/*	Static code analysis
Dalfox	v2.9.3	*https://github.com/hahwul/dalfox*	XSS Scanner
Gitleaks	8.21.2	*https://github.com/gitleaks/gitleaks*	Secret Scanning in source code
Docker / Docker Compose	23.0.1	*https://docs.docker.com/get-started/get-docker/*	Runtime and debug environment
AFLplusplus	4.31c	*https://aflplus.plus/*	Dynamic Testing: Fuzzing

2.4.3 Findings and Recommendations

2.4.3.1 Findings in QGIS Server

These findings were identified in the QGIS Server:

Component	Reference	Finding	Recommendations	Fixed Versions	Implemented Fix
QGIS Source Code	L1 - Low	**"Unpinned" Versions of Dependencies Used** QGIS relies on external dependencies in their code base. It was found that for both the C++ and Python dependencies, only the name of a dependency is specified. This means that the latest version of a specified package is used. The C++ dependencies are specified in `src/vcpkg/vcpkg.json` and only specify a minimal version (`baseline`) for two package sources. All other dependencies do not specify a version that should be installed. The Python dependencies defined in the `requirements.txt` only name the dependencies used without specifying which version should be installed. Using unpinned dependency versions can lead to unpredictable build failures, introduce breaking changes, or inadvertently incorporate new security vulnerabilities when dependencies are updated without verification. This also increases the risk of supply chain attacks.	To mitigate the risks associated with unpinned dependency versions, it is recommended to implement the following measures: • Use exact versions for all dependencies in your project's manifest files (`vckg.json` and `requirements.txt)` • If supported by the dependency manager, use a "lock" mechanism that uses a hash value to ensure the installed dependency matches the one specified in a lock file. • Regularly review and update dependencies: i.e. don't "pin and forget."	N/A	No changes necessary, as VCKG already pins the versions automatically.

2.4.3.2 Findings in QWC2

These findings were identified in QWC2:

Component	Reference	Finding	Recommendations	Fixed Versions	Implemented Fix
QWC2	H1 – High CVE-2025-11183	**Stored Cross-Site Scripting Vulnerability in Attribute Table** There is a Cross-Site Scripting (XSS) in the attribute table used to edit points and lines 1. Ensure you are logged in with a user that has edit permissions for the present attributes in the attributes table (Lines, Points and Polygons, etc). 2. Open the editing dialog via Burger Menu (top right) > Map Tools > Editing 3. Draw an arbitrary line (double click to finish line) 4. Open the attribute table by clicking the table button in the editing dialog 5. Edit the Name (or Description) of the line to this value: `<img onerror=alert(123) src=x>` 6. An alert message with the text 123 will pop up confirming the XSS vulnerability This allows adversaries with editing capabilities of an attribute layer to place and then execute arbitrary JavaScript code. This code will later be run when other users display the attribute layer. XSS Attacks are often used to steal credentials or login tokens of other users.	The following recommendations are suggested to reduce the risk of XSS vulnerabilities: • Use the framework's built-in templating functionalities to securely render content • Validate and sanitize all inputs on the backend • Use a CSP to disallow in-line JavaScript execution • Use context specific output encoding before rendering it Additional tips how XSS can be prevented can be found here: *https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html*	2025.08.14	User input is now sanitized before rendering it. All users are encouraged to update to the latest version.

QWC2 Registration	H2 – High CVE-2025-11184	**Stored Cross-Site Scripting Vulnerability in the Registration Module** There is a potential Cross-Site Scripting (XSS) in the registration module via registrable groups. 1. Ensure you are logged in with a user that has permission to make a registerable group. 2. Create a registrable group with the following description: `<script>alert(document.domain)</script>` 3. Navigate to `http://localhost:8088/registration/register` 4. An alert message with the domain name will pop up confirming the XSS vulnerability This allows adversaries with editing capabilities of registrable groups to place scripts that will execute arbitrary JavaScript code. This code will later be run when other users display the registration page. XSS Attacks are often used to steal credentials or login tokens of other users.	The following recommendations are suggested to reduce the risk of XSS vulnerabilities: • Use the framework's built-in templating functionalities to securely render content • Validate and sanitize all inputs on the backend • Use a CSP to disallow in-line JavaScript execution • Use context specific output encoding before rendering it Additional tips how XSS can be prevented can be found here: *https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html*	2025.09.30	User input is now encoded before rendering it. All users are encouraged to update to the latest version.

QWC2 Source Code	L2 – Low	**Bing API Key in Commit History** A Bing API token can be extracted from the Git history. The token is no longer part of the current source code, but it must have been committed at some point in the past and then removed. The token was introduced in the commit `833e30f` and can also be found here: *https://github.com/qgis/qwc2/blob/833e30fa0a59e6595feef2ad55b1f257218ba233/localConfig.json#L5* Note: The NTC did not verify the validity or permissions of this token. Exposed API tokens can potentially be used to access confidential data or consume cloud resources at the cost of the original owner.	Here are several recommendations aimed at minimizing the risk of exposed tokens: • Revoke the token if it is still valid • Monitor access and the usage of resources on the backend • Secrets should not be hardcoded in code repositories or CI/CD configuration files. Employ tools such as gitleaks or git-secrets to detect such secrets. OWASP has extensive documentation on Secrets Management: *https://cheatsheetseries.owasp.org/cheatsheets/Secrets_Management_Cheat_Sheet.html*	N/A	The Bing API key is obsolete / revoked.

QWC2 Admin	L3 - Low	**CSRF token cookie without secure flags** The CSRF token is transmitted via a cookie, but the cookie does not have the secure flag set. If the Secure flag is not set, the cookie can be transmitted over unencrypted HTTP connections. This exposes it to interception over insecure networks (e.g., public Wi-Fi). Even though CSRF tokens are a mitigation mechanism, improperly configured cookie attributes can weaken the overall protection and introduce new attack surfaces.	The following measures are recommended to address the issue: • Avoid storing CSRF tokens in cookies unless necessary. As an alternative, a hidden input could be used for example. More details on this can be found from *OWASP*. • Set the Secure flag on the CSRF token cookie to ensure transmission only over HTTPS • Enforce HTTPS across the entire application to ensure the Secure flag is effective	N/A	No patch is necessary as this can already be enabled using the `JWT_COOKIE_SECURE` flag in the docker compose file. Users are encouraged to set the flag to true.
QWC2 Source Code	I1 - Info	**Vulnerable Version of Axios Used** The tested version of QWC2 uses a version of Axios that is vulnerable to Server-Side Request Forgery. As specified in the `yarn.lock` file, the version `1.8.1` of the library is used. An initial review indicates that QWC2 is not affected by this vulnerability. The issue specifically targets axios clients (created using `axios.create`), for which no instances were found in the code base. More details about the vulnerability can be found here: *https://github.com/axios/axios/security/advisories/GHSA-jr5f-v2jv-69x6*	To ensure that future code changes do not introduce unforeseen vulnerabilities, it is recommended to update to an Axios version `>=1.8.2`.	v2025.08.14	Dependency was updated. All users are encouraged to update to the latest version of QWC2.

3 Appendix

3.1 Risk Categories

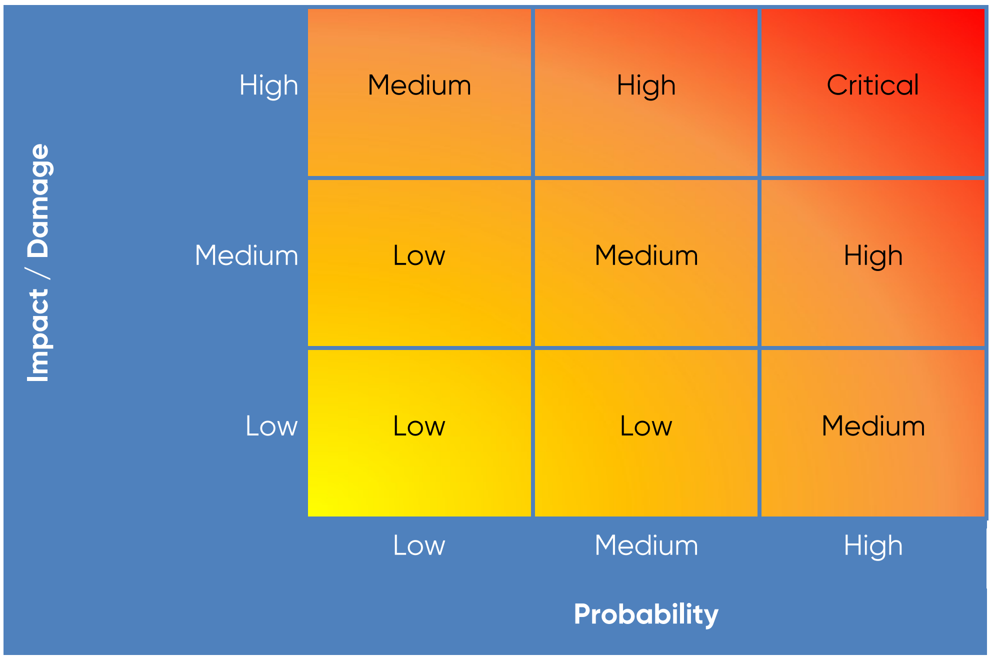
The findings in this report can be classified into these categories.

- Critical
- High
- Medium
- Low
- Info

3.1.1 Calculation of Risk Categories

To determine the applicable risk category, the following slightly simplified but widely used and practical formula is applied:

- Risk = Probability * Impact



Impact / Damage			
High	Medium	High	Critical
Medium	Low	Medium	High
Low	Low	Low	Medium
	Low	Medium	High
		Probability	

Findings that are not directly associated with a risk but whose recommended measures contribute to increasing the system's security level are categorized as **"Info"**.

3.1.2 Probability

The likelihood of occurrence is divided into three categories: High, Medium, and Low, and takes into account the following parameters from the DREAD Risk Assessment Model:

- **Exploitability** – How much effort is required to execute the attack?
- **Reproducibility** – How easy is it to replicate the attack?
- **Discoverability** – How easy is it to identify the vulnerability?

3.1.3 Impact

The impact is divided into three categories: High, Medium, and Low, and takes into account the following parameters from the DREAD Risk Assessment Model:

- **Damage** - How severe would a successful attack be?
- **Affected users**– How many users would be affected?

3.2 Definitions for Test Conditions

The graphic displaying the test conditions provides information about the conditions under which the tests were conducted. The individual categories and possible values are described below.

3.2.1 Attack Vector

The attack vector describes the method through which the tests were conducted / simulated.

- **Internet**
  The tests were conducted over the internet. It must be assumed that the identified vulnerabilities can be exploited by any attacker with internet access.
- **Local Network**
  The tests were conducted over the local network. This network access is typically used by employees, but also by partner companies, suppliers, or other insiders.
- **Local Access**
  For the tests, local access to the test system was provided to allow interaction not only with the network-exposed services but also with the operating system. Access is typically provided via SSH, Remote Desktop, Citrix, etc., and requires valid credentials.
- **Physical Access**
  During the tests, physical access to the system was also available. This allowed interaction not only over the network but also through other physical interfaces. For example, access to debugging ports, removal of memory chips or hard drives, replacement of SIM cards, etc., would be possible

3.2.2 Testing Approach

The approach describes the level of knowledge the testers have about the test system.

- **Blackbox**
  The testers have no prior information about the security policies, architecture, configuration, source code, etc., of the system being tested.
- **Greybox**
  The testers receive some preliminary information about the system being tested and have the option to request additional details from the system operator if needed.
- **Whitebox**
  The testers have access to all security-relevant information, including documentation, configurations, source code, etc.

3.2.3 Access permissions

The Access permissions describe the privileges granted to the testers.

- **Unprivileged**
  No credentials were provided, and the tests were conducted without user credentials or special permissions.
- **User**
  The credentials of a standard user without elevated privileges were used. The credentials were either provided by the operator or generated through a self-registration process.
- **Administrator**
  Credentials with elevated privileges (e.g., those of an administrator) were used. This allowed testing of functionalities that are not accessible to a standard user.

Note: Even if credentials with specific privileges are provided, tests are also conducted without these privileges or with reduced privileges. For example, it is assessed whether administrative functions can be used by standard or unprivileged users as well.

3.2.4 Degree of Automation

The degree of automation describes the balance between automated and manual testing.

- **Fully automated**
  The tests are primarily conducted in an automated manner using scanning tools such as Nessus. The results are manually verified to identify and eliminate false positives. However, complex vulnerabilities or those requiring an understanding of the application's logic may not always be detected.
- **Partly automated**
  The tests are conducted in an automated way and are completed by manual testing in areas deemed beneficial, based on the experience of the testers.
- **Manual**
  The tests are predominantly conducted manually by experienced security specialists. Where appropriate, automated tools are also used to optimize resources and leverage the testers' expertise in areas where automated tools reach their limitations.

3.2.5 Allowlisting

The allowlisting flag documents whether certain security measures were disabled or adjusted for the execution of the tests. This may be necessary in certain situations to complete the tests within a reasonable timeframe or to achieve a higher depth of testing:

- **Yes**
  Some security measures were disabled or adjusted in coordination with the operators. The details are provided in the comments.
- **No**
  No security measures were disabled.

3.2.6 Timeboxed

The Timeboxing flag indicates whether intentionally less time was allocated for the tests than would be required for a complete assessment.

- **Yes**
  The test is conducted using a timeboxed approach to achieve an optimal cost-benefit ratio. The tests are performed within a limited time frame and focus on the most likely security vulnerabilities ("low-hanging fruits"). This approach is particularly suitable for target systems where a security assessment is important, but resources are limited.
- **No**
  The test is conducted to the extent recommended by NTC to allow for a comprehensive assessment.