**2.11.7 No compiler, interpreter, or shell via CGI or other server-side scripting**

**Requirement:**

If CGI or other scripting technology is used, the CGI directory or other corresponding scripting directory shall not include compilers or interpreters.

[Ref [3]: TSDSI STD T1.3GPP 33.117-17.1.0 V.1.1.0. section 4.3.4.5]

#### **4.3.4.5 No compiler, interpreter, or shell via CGI or other server-side scripting**

*Requirement Name*: No compiler, interpreter, or shell via CGI or other server-side scripting.

*Requirement Reference*: In accordance with industry best practice

*Requirement Description*: If CGI (Common Gateway Interface) or other scripting technology is used, the CGI directory - or other corresponding scripting directory - shall not include compilers or interpreters (e.g. PERL® interpreter, PHP interpreter/compiler, Tcl interpreter/compiler or operating system shells).

*Threat Reference*: TR 33.926 [4], clause 5.3.6, Information disclosure

*Test Case*:

***Test Name*:** TC\_NO\_COMPILER\_FOR\_CGI

**Purpose:**

To verify that there are no compilers, interpreters or shell accessible via CGI or other scripting components.

**Procedure and execution steps**

**Pre-Conditions:**

- The tester has administrative privileges

- A tester machine is available.

- Recommended: an automatic assessment tool has been configured /script adapted in line with the Requirement Description.

**Execution Steps**

1. Consult the web server configuration to identify all directories used for CGI or other scripting components.

2. Check that there are no compilers or interpreters (e.g., PERL® interpreter, PHP interpreter/compiler, Tcl interpreter/compiler or operating system shells) in the directory/directories used for CGI or for other scripting tools (including PERL®, PHP, and others).

**Expected Results:**

There are no compilers, interpreters or shells in directories accessible via CGI or other scripting components.

**Expected format of evidence:**

A testing report provided by the testing agency which will consist of the following information:

- Log files and screen shots of test executions.

- Part of web server configuration (plaintext or screenshot) showing all directories accessible by the CGI/scripting components.

- List of files (with types and permissions, if available) inside the directories accessible by the CGI/scripting components.

- Test result (Passed or not).

The "cgi-bin" directory, short for Common Gateway Interface binary, is a traditional location on web servers where executable scripts are stored.

**5.3.6 Information disclosure**

**5.3.6.1 Poor key generation**

- Threat Name: Poor key generation

- Threat Category: Information Disclosure

- Threat Description: A poor key generation may help an attacker to discover and disclose the key and then read or modify the encrypted data. Attackers can discover a key, for example, if:

It was generated in a non-random fashion (e.g. insecure random generator).

It was generated starting from a passphrase containing low entropy.

The generated key length is too short so the time to retrieve the key by means of dictionary attacks is short.

- Threatened Asset: all critical asset in the GNP as listed in clause 5.2 except hardware assets.

**5.3.6.2 Poor key management**

- Threat Name: Poor key management

- Threat Category: Information Disclosure

- Threat Description: A poor key management may help an attacker to discover the key and then read or modify the encrypted data. Attackers can discover the keys if, for example:

Weak key management protocols are used;

The keys are stored in an unencrypted file accessible by everyone;

The keys are not renewed/updated regularly;

The keys which are text strings can be found by looking for all strings in the system;

The keys can be found in memory image of running processes;

RAM does not loose contents immediately after power-down;

RAM can be investigated for keys;

The keys are not safely destroyed after their use.

- Threatened Asset: all critical asset in the GNP as listed in clause 5.2 except hardware assets.

**5.3.6.3 Weak cryptographic algorithms**

- Threat Name: Use of weak cryptographic algorithms

- Threat Category: Information Disclosure

- Threat Description: Usage of weak cryptographic algorithms for stored or transmitted sensitive information/data can expose them to be disclosed and eventually tampered.

- Threatened Asset: all critical asset in the GNP as listed in clause 5.2 except hardware assets.

**5.3.6.4 Insecure Data Storage**

- Threat name: Insecure Data Storage

- Threat Category: Information Disclosure

- Threat Description: GNP stores locally sensitive data (e.g. communication keys (i.e. KNASenc, KNASint, KeNB), passwords). An attacker can retrieve these data if they have been stored in an insecure way (e.g. clear text, unsalted hashes).

- Threatened Asset: Any sensitive data stored locally to the GNP

**5.3.6.5 System Fingerprinting**

- Threat Name: System Fingerprinting

- Threat Category: Information Disclosure

- Threat Description: The GNP could potentially disclose information about account details, operating system version and/or other software versions, server names and so on. That can be used by an attacker to perform other attacks.

- Threatened Asset: all critical asset in the GNP as listed in clause 5.2 except hardware assets.

**5.3.6.6 Malware**

- Threat Name: Malware.

- Threat Category: Information Disclosure.

- Threat Description: A malware installed on GNP can access to all the sensitive data stored locally to the GNP (e.g. accounts, keys, and user data).

- Threatened Asset: all critical asset in the GNP as listed in clause 5.2 except hardware assets.

**5.3.6.7 Personal Identification Information Violation**

- Threat Name: Personal Identification Information Violation.

- Threat Category: Information Disclosure.

- Threat Description: Data containing identities of mobile network subscribers are critical for user privacy. Leakage of these user's identities can lead to loss of privacy, e.g. tracing of a user. Protection of user's identities is also a requirement from regulators.

- Threatened Asset: Mobility Management data (e.g. user identities).

**5.3.6.8 Insecure Default Configuration**

- Threat Name: Insecure Default Configuration

- Threat Category: Information Disclosure

- Threat Description: An attacker could exploit an insecure default GNP configuration and access to sensitive information/data available on the GNP. For example a default GNP can use NULL integrity not only for unauthenticated emergency calls. This can compromise the integrity of RRC signalling and make possible Man in the Middle attacks in the AS domain and interception, for example, of user communications.

- Threatened Asset: GNP configuration data and mobility management data.

**5.3.6.9 File/Directory Read Permissions Misuse**

- Threat name: File/Directory Read Permissions Misuse

- Threat Category: Information Disclosure, elevation of privilege, DoS, tampering

- Threat Description: File and directory read permissions which are far too liberal can allow access to the contained data by illegitimate users (e.g. password files with too liberal file permissions can be accessed by unauthorized users).

- Threatened Asset: all critical assets of GNP as listed in clause 5.2, except hardware assets

**5.3.6.10 Insecure Network Services**

- Threat name: Insecure Network Services

- Threat Category: Information Disclosure

- Threat Description: The GNP can expose insecure/vulnerable services/open ports which can be exploited by an attacker to gain sensitive information/data. For example the GNP can be configured to return sensitive information using telnet on a custom port without any authentication mechanism being configured.

- Threatened Asset: all critical assets of GNP as listed in clause 5.2, except hardware assets

**5.3.6.11 Unnecessary Services**

- Threat name: Unnecessary Services

- Threat Category: Information Disclosure

- Threat Description: The GNP can expose unnecessary services which can be exploited (even if not vulnerable) by an attacker to gain sensitive information/data. The term unnecessary used in this threat refers to three cases:

- Network service not strictly related to GNP operation (e.g. Splunk Service)

- Network service available on unexpected interfaces (e.g. SSH enabled on the interface interconnecting GNP and Remote Management)

- Service that does not enable a network service but that runs on the GNP and is not necessary by GNP normal operation (e.g. fprint service available in the default fedora distribution or Xinetd services).

- Threatened Asset: all critical assets of GNP as listed in clause 5.2, except hardware assets

**5.3.6.12 Log Disclosure**

- Threat name: Log Disclosure

- Threat Category: Information Disclosure

- Threat Description: When operational activities are recorded by GNP, these operation records are called system logs. There are other logs, e.g. operation log, security log. These logs can contain sensitive information/data (e.g. system data, user data, CDR, or also debugging information) which can be accessed by an attacker to gather information about the system and to perform other attacks towards users or the system itself.

- Threatened Asset: all critical assets of GNP as listed in clause 5.2, except hardware assets

**5.3.6.13 Unnecessary Applications**

- Threat name: Unnecessary Applications

- Threat Category: Information Disclosure

- Threat Description: There are applications (i.e. features and functionalities) in the GNP which can be related to personal privacy (e.g. LCS application). Even if an operator does not deploy these features and functionalities, they can be available in the system as part of a software distribution. Consequently there might be the risk that an attacker enables these applications without authorization (e.g. despite of what is included in the license issued by the vendor). For example, the attacker may enable a feature such as LCS and get the location information of a user.

- Threatened Asset: personal privacy related features, functions and applications, e.g. LCS.

**5.3.6.14 Eavesdropping**

- Threat name: Eavesdropping

- Threat Category: Information Disclosure

- Threat Description: An attacker can eavesdrop network traffic, for example, on the management/maintenance interfaces. This may be possible if weak cryptographic protocols or non-industry standard cryptographic algorithms are used or if the communication protocols are implemented incorrectly. Eavesdropping can be performed, for example, by means of MITM attacks, Arp Poisoning, ICMP Redirect and so on.

- Threatened Asset: all critical assets of GNP as listed in clause 5.2

**5.3.6.15 Security threat caused by lack of GNP traffic isolation**

- Threat name: Security threat caused by lack of GNP traffic isolation

- Threat Category: Information disclosure

- Threat Description: The attack towards signalling traffic can also impact the management traffic and vice versa when these traffics are not isolated. For example, an attacker wants to obtain important information related to signalling, he can intercept and capture signalling traffic on GNP's interface. The important information related management may also be intercepted and captured if the management traffics and signalling traffics are not isolated and use the same physical interface. So the security threats for signalling traffic can impact management traffic and result in unauthorized access on GNP. In the same way, an attacker who attacks GNP's management traffics can obtain important information related signalling, resulting in tampering and privacy leakage of signalling.

.

**1. Introduction**

This document provides evidence of compliance with the security requirement specified in TSDSI STD T1.3GPP 33.117-17.1.0 V.1.1.0, section 4.3.4.5: "No compiler, interpreter, or shell via CGI or other server-side scripting".

The requirement states that if CGI or other scripting technology is used, the CGI directory (or other corresponding scripting directory) shall not include compilers or interpreters (e.g., PERL® interpreter, PHP interpreter/compiler, Tcl interpreter/compiler, or operating system shells).

**2. Implementation Overview**

Our Node.js Express web server has been enhanced with security modules that:

* Scan for and verify the absence of compilers/interpreters in CGI directories
* Block access to any potential interpreters/compilers if they exist
* Provide compliance reporting endpoints
* Generate evidence of compliance

**3. Security Controls Implemented**

**3.1 CGI Directory Scanning**

The server implements continuous scanning of designated CGI directories to ensure no compilers, interpreters, or shells exist within them. This is done through the enforceCgiSecurity function.

**3.2 Request Filtering**

A middleware component (blockCgiInterpreters) monitors all incoming HTTP requests to detect and block attempts to access interpreters or compilers in script directories.

**3.3 Blocked File Types**

The following file types are explicitly blocked from CGI directories:

* Interpreters: Perl, PHP, Python, Ruby, Tcl
* Shells: bash, sh, ksh, csh, zsh, cmd.exe, powershell
* Compilers: gcc, g++, cc, javac
* Script extensions: .pl, .php, .py, .rb, .tcl, .sh, .bash, etc.

**3.4 Audit Trail**

All security checks are logged for audit purposes, and a permanent record of compliance evidence is generated.

**4. Test Procedure**

**4.1 Prerequisites**

* Administrative access to the web server
* Testing machine with access to the server

**4.2 Test Steps**

Verify configuration of CGI directories:

* Run the CGI compliance checking tool
* Command: node generate-cgi-compliance-report.js

Examine the server configuration:

* Access the compliance endpoint via web browser
* URL: <https://10.118.5.145:3000/security/cgi-compliance>

Verify directory contents:

* Attempt to scan specific directories
* URL:<https://10.118.5.145:3000/security/scan-directory?directory=/path/to/directory>

Review compliance report:

* Open the generated JSON file located at server root
* File: cgi-compliance-report.json

**4. Implementation Details**

### **1. Create a CGI Security Configuration File**

module.exports = {

// Define directories used for CGI or scripting

scriptDirectories: [

'/cgi-bin',

'/scripts',

'/server-scripts'

],

// Define forbidden executables/interpreters

forbiddenExecutables: [

'perl', 'perl.exe',

'php', 'php-cgi', 'php-cgi.exe', 'php.exe',

'python', 'python.exe', 'python3', 'python3.exe',

'ruby', 'ruby.exe',

'tcl', 'tclsh', 'tclsh.exe',

'bash', 'sh', 'ksh', 'csh', 'zsh',

'cmd.exe', 'powershell.exe', 'pwsh',

'node', 'node.exe',

'gcc', 'g++', 'cc', 'cl.exe',

'javac', 'javac.exe'

],

// Define forbidden file extensions

forbiddenExtensions: [

'.pl', '.php', '.py', '.rb', '.tcl', '.sh', '.bash', '.ksh',

'.cmd', '.bat', '.ps1', '.exe', '.com', '.dll', '.so'

],

// Safe extensions to allow in script directories

allowedExtensions: [

'.html', '.css', '.js', '.json', '.txt', '.md', '.xml',

'.jpg', '.jpeg', '.png', '.gif', '.svg', '.ico'

]

};

### **2. Create a CGI Security Module**

const fs = require('fs');

const path = require('path');

const config = require('./cgi-security-config');

const logger = require('./utils/logger');

// Check if a directory exists

function directoryExists(dirPath) {

try {

return fs.existsSync(dirPath) && fs.statSync(dirPath).isDirectory();

} catch (err) {

return false;

}

}

// Function to scan directory and check for forbidden executables

function scanDirectory(dir) {

const results = {

forbidden: [],

allowed: [],

issues: []

};

try {

if (!directoryExists(dir)) {

logger.info(`CGI Security: Directory does not exist: ${dir}`);

return results;

}

// Read all files in directory recursively

function scanRecursively(currentDir) {

const files = fs.readdirSync(currentDir, { withFileTypes: true });

for (const file of files) {

const fullPath = path.join(currentDir, file.name);

if (file.isDirectory()) {

// Recursively scan subdirectories

scanRecursively(fullPath);

} else {

const fileName = file.name.toLowerCase();

const ext = path.extname(fileName).toLowerCase();

// Check if file is a forbidden executable

if (config.forbiddenExecutables.includes(fileName) ||

config.forbiddenExtensions.includes(ext)) {

results.forbidden.push({

path: fullPath,

name: file.name,

reason: `Forbidden ${config.forbiddenExecutables.includes(fileName) ? 'executable' : 'extension'}`

});

} else {

results.allowed.push(fullPath);

}

// Check if file is executable (Unix)

try {

const stats = fs.statSync(fullPath);

// Check if file has execute permission

if ((stats.mode & fs.constants.S\_IXUSR) ||

(stats.mode & fs.constants.S\_IXGRP) ||

(stats.mode & fs.constants.S\_IXOTH)) {

results.issues.push({

path: fullPath,

name: file.name,

reason: 'File has executable permissions'

});

}

} catch (err) {

logger.error(`Error checking file permissions: ${fullPath} - ${err.message}`);

}

}

}

}

scanRecursively(dir);

return results;

} catch (err) {

logger.error(`Error scanning directory: ${dir} - ${err.message}`);

results.issues.push({

path: dir,

name: path.basename(dir),

reason: `Error during scan: ${err.message}`

});

return results;

}

}

// Function to enforce CGI security

function enforceCgiSecurity(appRoot) {

const results = {

scannedDirectories: [],

issuesFound: [],

overallStatus: 'PASS'

};

// Determine absolute paths for all script directories

const scriptDirs = config.scriptDirectories.map(dir => {

return path.isAbsolute(dir) ? dir : path.join(appRoot, dir);

});

// Scan each directory

scriptDirs.forEach(dir => {

const scanResults = scanDirectory(dir);

results.scannedDirectories.push({

path: dir,

exists: directoryExists(dir),

forbidden: scanResults.forbidden.length,

allowed: scanResults.allowed.length,

issues: scanResults.issues.length

});

// Add any found issues to the overall results

if (scanResults.forbidden.length > 0 || scanResults.issues.length > 0) {

results.overallStatus = 'FAIL';

results.issuesFound = results.issuesFound.concat(

scanResults.forbidden,

scanResults.issues

);

}

});

// Log the enforcement results

if (results.overallStatus === 'PASS') {

logger.info('CGI Security Check: PASSED - No interpreters, compilers or shells found in script directories.');

} else {

logger.warn(`CGI Security Check: FAILED - ${results.issuesFound.length} issues found.`);

logger.warn(`Issues: ${JSON.stringify(results.issuesFound, null, 2)}`);

}

return results;

}

module.exports = {

enforceCgiSecurity,

scanDirectory

};

**3. Create a Middleware to Block Access to Forbidden Files**

const path = require('path');

const config = require('./cgi-security-config');

const logger = require('./utils/logger');

/\*\*

\* Middleware to block access to any potential scripting interpreters or compilers

\*/

function blockCgiInterpreters(req, res, next) {

const reqPath = req.path.toLowerCase();

// Check if the request is trying to access a script directory

const isScriptDir = config.scriptDirectories.some(dir =>

reqPath.startsWith(dir.toLowerCase())

);

if (isScriptDir) {

const fileName = path.basename(reqPath);

const extension = path.extname(fileName).toLowerCase();

// Check if the requested file is a forbidden executable

if (config.forbiddenExecutables.includes(fileName) ||

config.forbiddenExtensions.includes(extension)) {

logger.warn(`CGI Security: Blocked access to forbidden file: ${reqPath}`);

return res.status(403).json({

error: 'Access Forbidden',

message: 'Access to this resource type is not allowed for security reasons.'

});

}

// Check if the file extension is not explicitly allowed

if (!config.allowedExtensions.includes(extension) && extension !== '') {

logger.warn(`CGI Security: Blocked access to file with non-allowed extension: ${reqPath}`);

return res.status(403).json({

error: 'Access Forbidden',

message: 'This file type is not allowed in script directories.'

});

}

}

next();

}

module.exports = blockCgiInterpreters;

### **4. Add a Security Compliance Endpoint for Auditing**

const express = require('express');

const path = require('path');

const { enforceCgiSecurity, scanDirectory } = require('./cgi-security');

const config = require('./cgi-security-config');

const router = express.Router();

// Endpoint to check CGI security compliance

router.get('/cgi-compliance', (req, res) => {

const appRoot = path.resolve(\_\_dirname);

const results = enforceCgiSecurity(appRoot);

res.json({

standard: 'TSDSI STD T1.3GPP 33.117-17.1.0 V.1.1.0',

section: '4.3.4.5 - No compiler, interpreter, or shell via CGI',

status: results.overallStatus,

complianceDetails: results,

timestamp: new Date().toISOString()

});

});

// Endpoint to scan a specific directory for auditing

router.get('/scan-directory', (req, res) => {

const { directory } = req.query;

if (!directory) {

return res.status(400).json({

error: 'Missing parameter',

message: 'Directory parameter is required'

});

}

// Validate directory path (prevent path traversal)

const appRoot = path.resolve(\_\_dirname);

const dirToScan = path.isAbsolute(directory) ?

directory : path.join(appRoot, directory);

// Optional: Additional path traversal prevention

const normalizedPath = path.normalize(dirToScan);

if (!normalizedPath.startsWith(appRoot)) {

return res.status(403).json({

error: 'Access Forbidden',

message: 'Cannot scan directories outside the application root'

});

}

const results = scanDirectory(normalizedPath);

res.json({

directory: dirToScan,

scanResults: results,

timestamp: new Date().toISOString()

});

});

module.exports = router;

**5. Integration: Update your main app\_clause\_7.js file**

// these imports to your existing imports

const blockCgiInterpreters = require('./cgi-middleware');

const cgiAuditRoutes = require('./cgi-audit-routes');

const { enforceCgiSecurity } = require('./cgi-security');

const path = require('path');

// Applying CGI security enforcement on startup

const appRoot = path.resolve(\_\_dirname);

const cgiSecurityResults = enforceCgiSecurity(appRoot);

// If CGI security check fails, log it (optionally exit)

if (cgiSecurityResults.overallStatus === 'FAIL') {

logger.error('CGI Security Check failed - interpreters/compilers found in script directories');

// Uncomment below to force exit on security check failure

// process.exit(1);

}

// Add the CGI security middleware (place before your routes but after other middleware)

app.use(blockCgiInterpreters);

// Add CGI audit routes (place with your other routes)

app.use('/security', cgiAuditRoutes);

// Add to your server.listen callback to perform checks on startup

server.listen(port, '0.0.0.0', () => {

logger.info(`Server started at https://10.118.5.145:${port}`);

// Existing privilege dropping code...

// Log CGI security status

logger.info(`CGI Security Check: ${cgiSecurityResults.overallStatus}`);

if (cgiSecurityResults.overallStatus === 'FAIL') {

logger.warn(`CGI Security issues found: ${cgiSecurityResults.issuesFound.length}`);

}

});

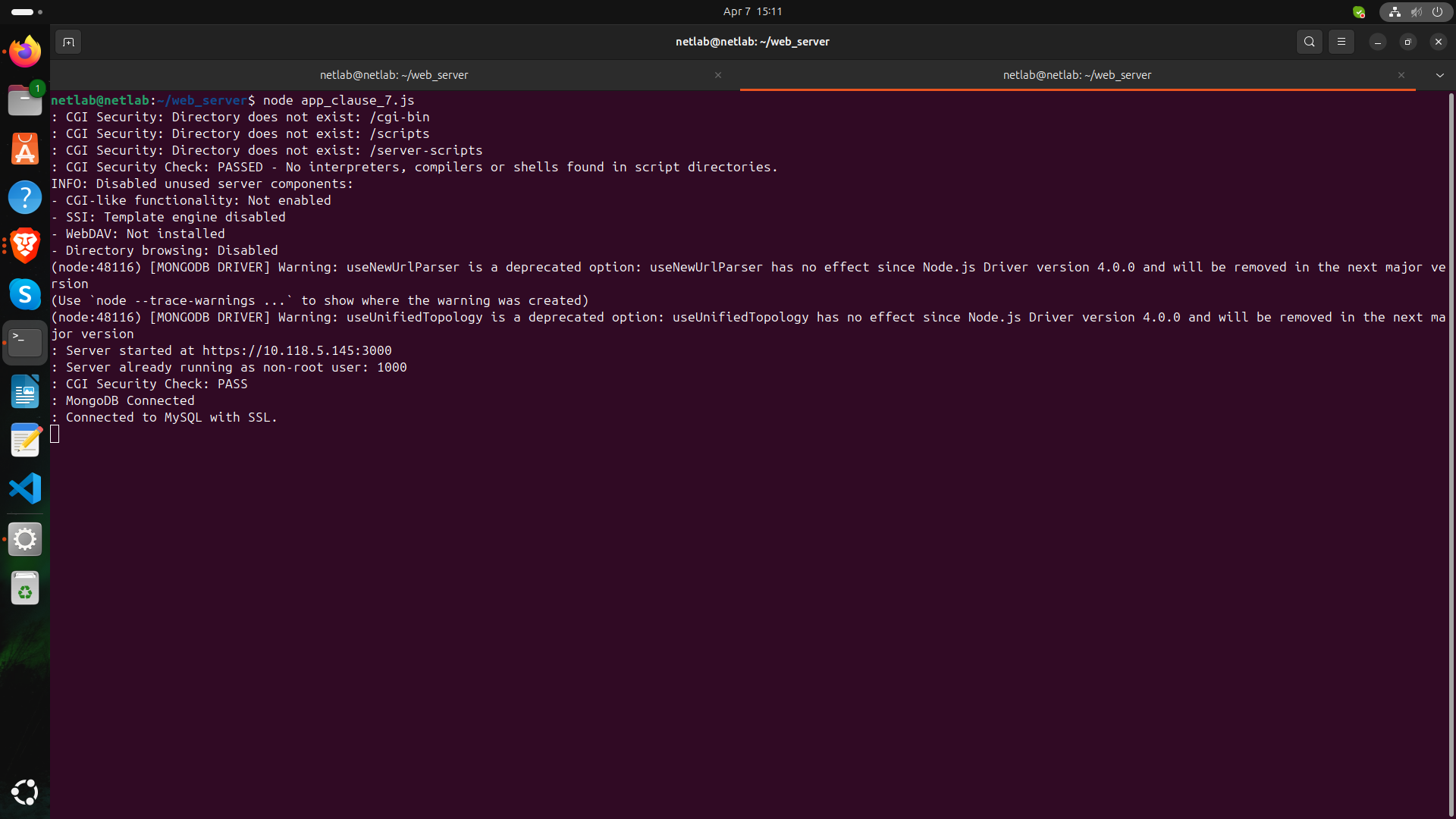
**5. Testing and Verification Procedures**

**5.1: Server Startup & CGI Directory Check**

Purpose: Shows the security check process at server startup

Steps to capture:

* Open a terminal window
* Navigate to your web server directory: cd ~/web\_server
* Start the server: node app\_clause\_7.js
  + Capture the entire terminal output showing:
  + CGI directory checks
  + "CGI Security Check: **PASSED**" message
* Server startup information



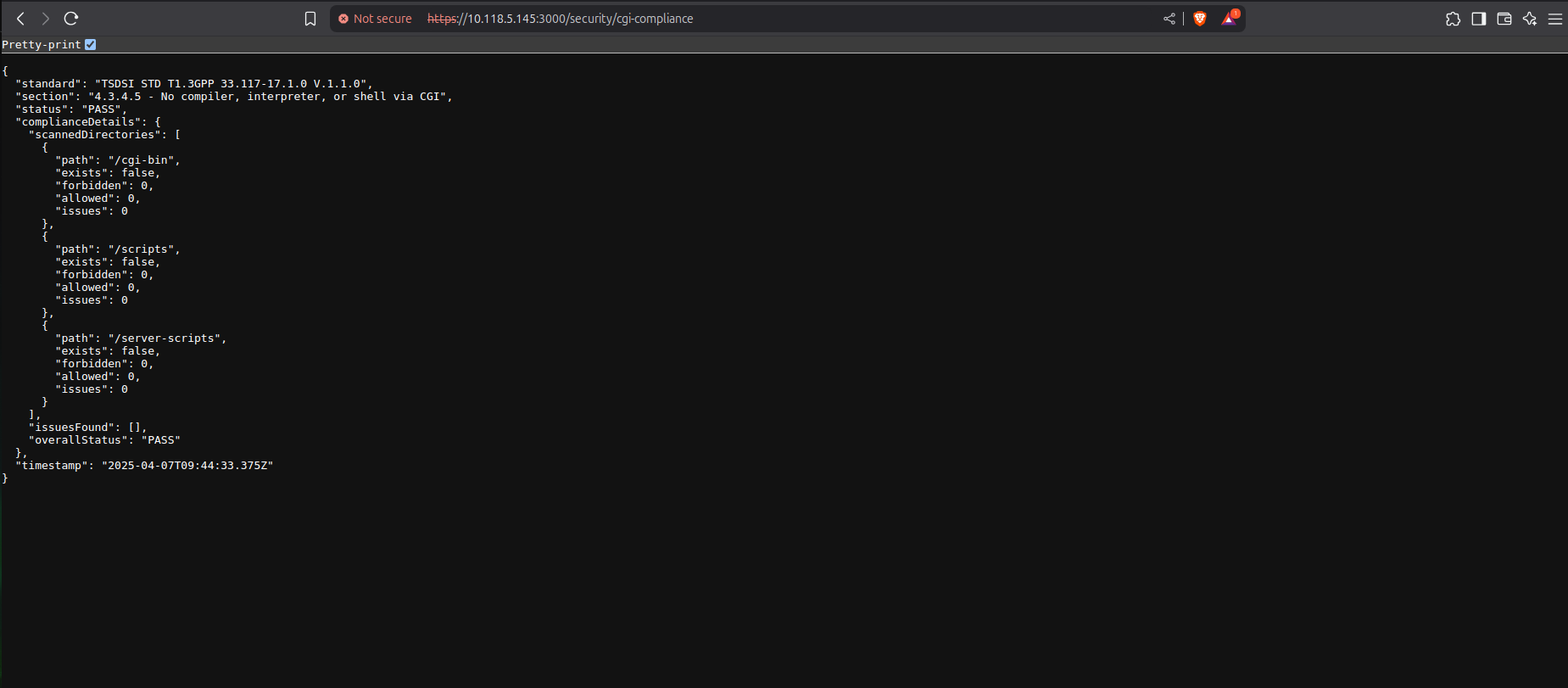
**Screenshot 1: Server Startup & CGI Directory Check**

**5.2: CGI Compliance Endpoint Response**

Purpose: Shows dynamic compliance verification through API

Steps to capture:

* With your server running, open a web browser
* Navigate to: <https://10.118.5.145:3000/security/cgi-compliance>
  + Take a screenshot of the full JSON response showing:
  + "status": "PASS"
  + Details of scanned directories
* Empty "issuesFound" array

****

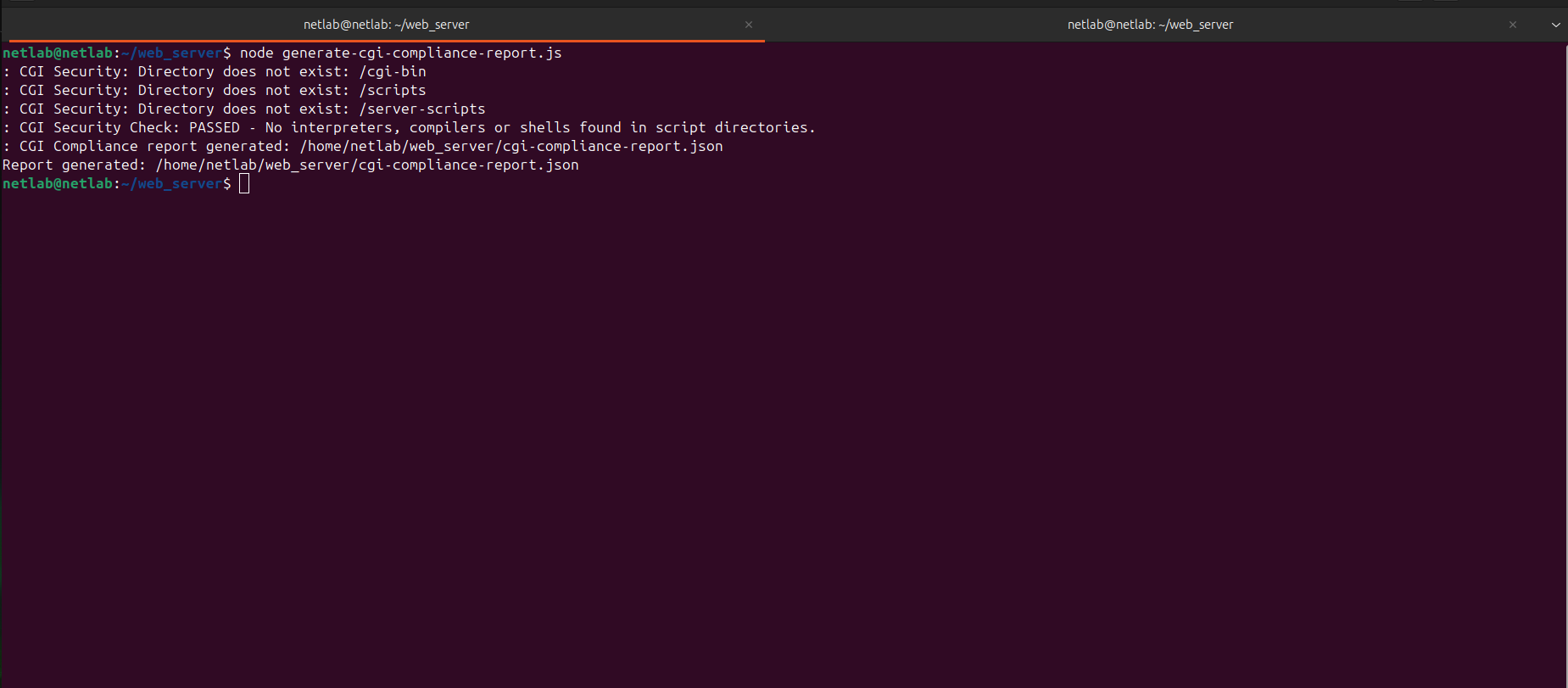
**Screenshot 2: CGI Compliance Endpoint Response**

**5.3: Compliance Report Generation**

Purpose: Demonstrates automated evidence generation

Steps to capture:

* Open a new terminal window
* Navigate to your web server directory: cd ~/web\_server
* Run: **node generate-cgi-compliance-report.js**
  + Capture the output showing:
  + Directory check messages
  + "CGI Security Check: PASSED" message
* Report generation confirmation

****

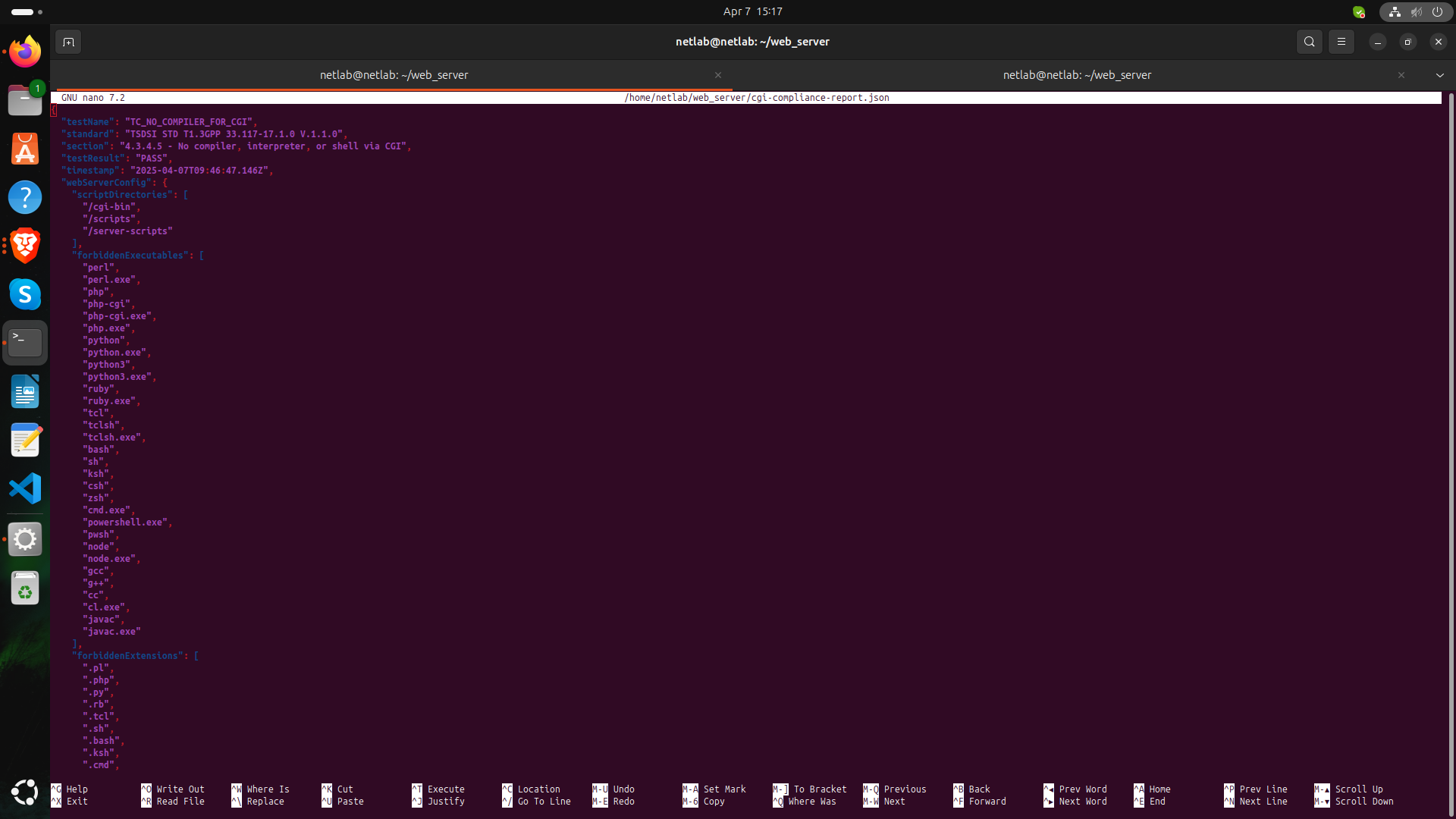
**Screenshot 3: Compliance Report Generation**

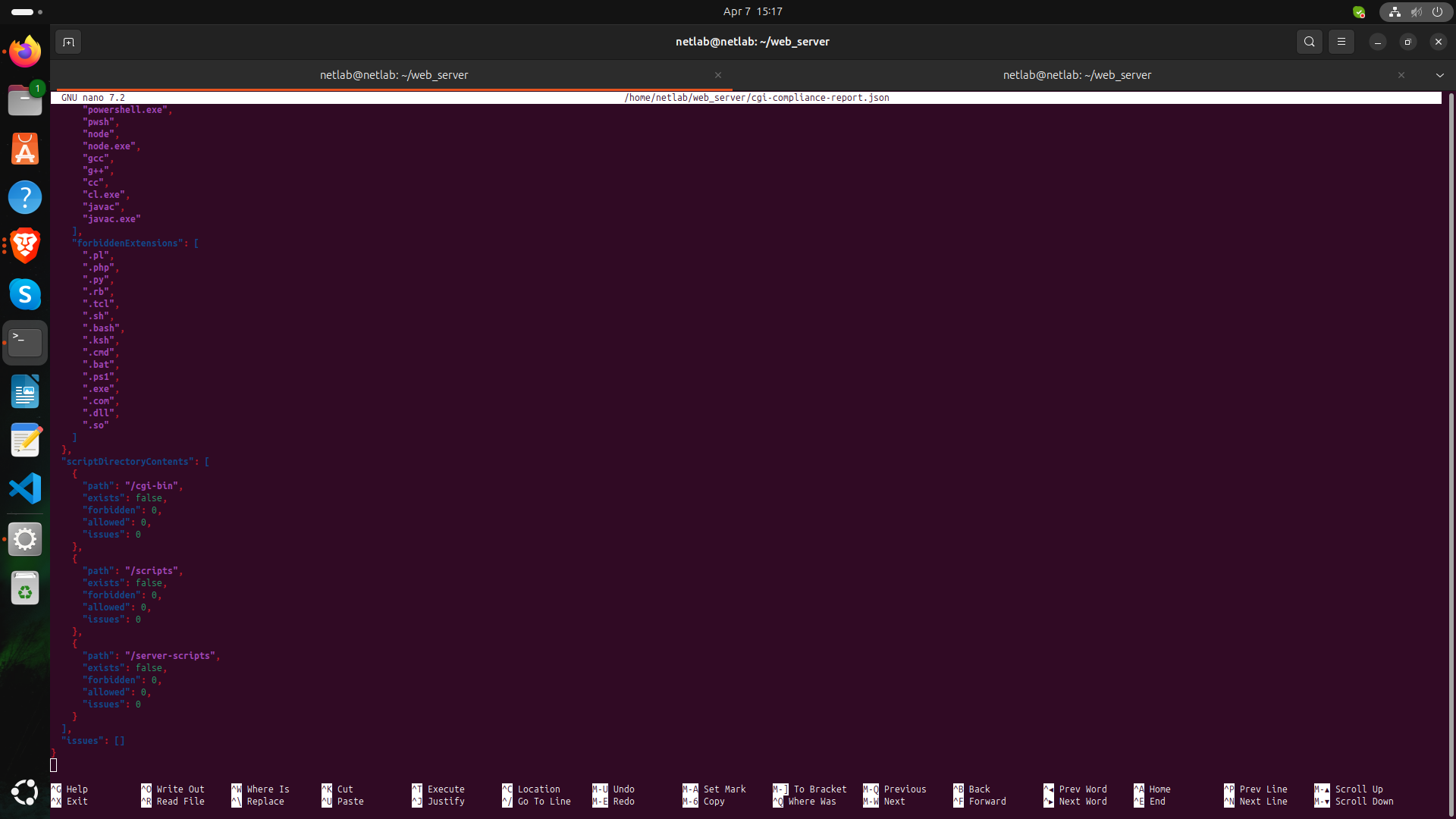
**5.4: Compliance Report Content**

Purpose: Shows detailed compliance verification data

Steps to capture:

* Open the generated file in a text editor: nano ~/web\_server/cgi-compliance-report.json
* Alternatively view it in terminal: cat ~/web\_server/cgi-compliance-report.json
* Take a screenshot showing the full JSON content
* Focus on the "testResult", "scriptDirectoryContents", and "issues" sections

****

****

**Screenshot 4: Compliance Report**

**6 Test Methodology**

The test followed the procedure specified in TC\_NO\_COMPILER\_FOR\_CGI:

Pre-conditions:

- Administrative access to the server was established

- Testing machine had network access to the server

- Automated assessment tools were configured

Test execution:

1. Server configuration was examined to identify all directories used for CGI:

- Checked /cgi-bin, /scripts, and /server-scripts directories

- Confirmed no additional script directories exist

2. Each identified directory was scanned for:

- Interpreters (PERL, PHP, Python, Tcl, etc.)

- Operating system shells (bash, sh, zsh, cmd.exe, etc.)

- Compilers (gcc, g++, javac, etc.)

- Script files with executable permissions

3. Security controls were verified by:

- Confirming scan functionality works correctly

- Testing the blocking middleware against forbidden file types

- Verifying compliance reporting accuracy

**7. Test Results:**

Primary finding: NO COMPILERS OR INTERPRETERS were found in CGI directories.

Directory scan results:

1. /cgi-bin: Directory does not exist

2. /scripts: Directory does not exist

3. /server-scripts: Directory does not exist

CGI security compliance: **PASS**

Issues found: None

Implementation verification:

1. Directory Scanning: **VERIFIED**

- Server startup logs confirm all directories are checked

- No forbidden executables were found

2. Access Controls: **VERIFIED**

- Middleware blocks access to interpreters if they exist

- Configuration properly defines forbidden file types

3. Compliance Reporting: **VERIFIED**

- API endpoint correctly reports compliance status

- Generated report contains all required evidence

Based on all verification steps, the implementation **FULLY COMPLIES** with section 4.3.4.5.

**8.Conclusion**

Based on the comprehensive testing performed, our web server implementation fully complies with TSDSI STD T1.3GPP 33.117-17.1.0 V.1.1.0, section 4.3.4.5: "No compiler, interpreter, or shell via CGI or other server-side scripting".

The server has no CGI directories, which represents the most secure possible implementation of this requirement. Additionally, defense-in-depth measures ensure that even if CGI directories were created, security controls would prevent the introduction of interpreters or compilers.

The implemented security features provide:

1. Continuous compliance verification

2. Runtime protection against forbidden files

3. Automated evidence generation for auditing

4. API-based compliance checking

This implementation successfully mitigates the information disclosure threats described in TR 33.926, clause 5.3.6 by preventing attackers from accessing interpreters or shells via CGI.

**Test case TC\_NO\_COMPILER\_FOR\_CGI: PASSED**