# Eclipse Temurin

Security Assessment

**June 14, 2024**

*Prepared for:*
**Stewart Addison**
The Eclipse Foundation
Organized by the Open Source Technology Improvement Fund, Inc.

*Prepared by:* **Sam Alws and Matt Schwager**

## About Trail of Bits

Founded in 2012 and headquartered in New York, Trail of Bits provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 100+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel.

We maintain an exhaustive list of publications at https://github.com/trailofbits/publications, with links to papers, presentations, public audit reports, and podcast appearances.

In recent years, Trail of Bits consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon.

We specialize in software testing and code review projects, supporting client organizations in the technology, defense, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom.

Trail of Bits also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash.

To keep up to date with our latest news and announcements, please follow @trailofbits on Twitter and explore our public repositories at https://github.com/trailofbits. To engage us directly, visit our "Contact" page at https://www.trailofbits.com/contact, or email us at info@trailofbits.com.

**Trail of Bits, Inc.**
497 Carroll St., Space 71, Seventh Floor
Brooklyn, NY 11215
https://www.trailofbits.com
info@trailofbits.com

## Notices and Remarks

### Copyright and Distribution

### Test Coverage Disclaimer

All activities undertaken by Trail of Bits in association with this project were performed in accordance with a statement of work and agreed upon project plan.

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

Trail of Bits uses automated testing techniques to rapidly test the controls and security properties of software. These techniques augment our manual security review work, but each has its limitations: for example, a tool may not generate a random edge case that violates a property or may not fully complete its analysis during the allotted time. Their use is also limited by the time and resource constraints of a project.

# Table of Contents

# Project Summary

## Contact Information

The following project manager was associated with this project:

**Jeff Braswell**, Project Manager
jeff.braswell@trailofbits.com

The following engineering director was associated with this project:

**Anders Helsing**, Engineering Director, Application Security
anders.helsing@trailofbits.com

The following consultants were associated with this project:

**Sam Alws**, Consultant
sam.alws@trailofbits.com

**Matt Schwager**, Consultant
matt.schwager@trailofbits.com

## Project Timeline

The significant events and milestones of the project are listed below.

| Date | Event |
|---|---|
| **December 1, 2023** | Pre-project kickoff call |
| **December 11, 2023** | Status update meeting #1 |
| **December 15, 2023** | Delivery of report draft |
| **December 15, 2023** | Report readout meeting |
| **June 14, 2024** | Delivery of comprehensive report |

# Executive Summary

## Engagement Overview

OSTIF engaged Trail of Bits to review the security of the Eclipse Foundation's Temurin project. The Temurin project is part of the top-level project Adoptium, and provides code and processes that support the building of quality Java runtime binaries and associated technologies that are high performance, enterprise caliber, cross platform, open-source licensed, and secure. At the highest level, Temurin takes source code for the implementation of Java SE versions from OpenJDK, builds and tests the code across a number of platform architectures, and makes the results available to end users in a wide variety of consumable formats.

A team of two consultants conducted the review from December 4 to December 15, 2023, for a total of four engineer-weeks of effort. Our testing efforts focused on authentication and authorization, data flow, and command injection vulnerabilities. With full access to source code and documentation, we performed static and dynamic testing of the codebase, using automated and manual processes.

During the audit, we also developed a set of Semgrep rules, which will be provided alongside this report in a zip file.

## Observations and Impact

We found a number of issues in which downloads (mainly software downloads) are performed without proper verification (TOB-TEMURIN-3, TOB-TEMURIN-4, TOB-TEMURIN-5, TOB-TEMURIN-6, TOB-TEMURIN-9, TOB-TEMURIN-12, TOB-TEMURIN-15, TOB-TEMURIN-16). We also found that the GitHub bot responsible for checking dependencies in the `infrastructure` repository is not configured correctly, preventing out-of-date dependencies from being detected (TOB-TEMURIN-19). We also noticed that dependencies are installed in many places in the `temurin-build` repository in an ad hoc manner, making it difficult to determine the full list of dependencies being used.

We found two high-severity issues that allow privileged users to perform code injection attacks on Jenkins build machines and Vagrant virtual machines (VMs) (TOB-TEMURIN-1, TOB-TEMURIN-17). We found two other high-severity issues involving vulnerabilities to person-in-the-middle attacks, through the API server's connection to its MongoDB database (TOB-TEMURIN-7) and SSH connections to a Nagios instance (TOB-TEMURIN-13).

## Recommendations

Based on the codebase maturity evaluation and findings identified during the security review, Trail of Bits recommends that the Eclipse Foundation take the following steps:

- **Remediate the findings disclosed in this report.** These findings should be addressed as part of a direct remediation or as part of any refactor that may occur when addressing other recommendations.

- **Create a centralized list of all the code locations where Temurin adds external dependencies.** Currently, Temurin adds dependencies throughout multiple Dockerfiles, Bash scripts, Ansible playbooks, and so on. This is especially true in the `temurin-build` repository, where it is very difficult to track down all the places where binaries are downloaded and run. We recommend making a single piece of documentation (or one piece of documentation per repository) containing a list of filenames and line numbers where dependencies are added.

## Finding Severities and Categories

The following tables provide the number of findings by severity and category.

### EXPOSURE ANALYSIS

| *Severity* | *Count* |
|---|---|
| High | 8 |
| Medium | 1 |
| Low | 4 |
| Informational | 5 |
| Undetermined | 1 |

### CATEGORY BREAKDOWN

| *Category* | *Count* |
|---|---|
| Access Controls | 3 |
| Configuration | 2 |
| Cryptography | 8 |
| Data Exposure | 1 |
| Data Validation | 2 |
| Patching | 3 |

## Project Goals

The engagement was scoped to provide a security assessment of the Eclipse Foundation's Temurin project. Specifically, we sought to answer the following non-exhaustive list of questions:

- How and where is data stored?
- How do users authenticate to the application(s)?
- How do internal systems authenticate to each other?
- How does user input flow through the system?
- How is the infrastructure managed?
- How does the system use cryptography?
- How does the system download and install software?
- What types of users or privileged parties exist in the system?
- Does the system interact with external services?
- Where do production workloads run?

## Project Targets

The engagement involved a review and testing of the targets listed below.

### api.adoptium.net

| | |
|---|---|
| Repository | https://github.com/adoptium/api.adoptium.net |
| Version | `52be774c47a374cd0cf13c40f2eb28f4b1158a16` |
| Type | Kotlin |
| Platform | Server |

### ci-jenkins-pipelines

| | |
|---|---|
| Repository | https://github.com/adoptium/ci-jenkins-pipelines |
| Version | `7b9559ce88321ff8111180fdc58421a0f9eadcef` |
| Type | Groovy |
| Platform | Server |

### infrastructure

| | |
|---|---|
| Repository | https://github.com/adoptium/infrastructure |
| Version | `9f6e77549a67031bea07efae3030942729baa186` |
| Type | Various scripting languages; Ansible playbooks |
| Platform | Server |

### jenkins-helper

| | |
|---|---|
| Repository | https://github.com/adoptium/jenkins-helper |
| Version | `3e12d3e25fe100e62275656342ee3f5396abb55e` |
| Type | Groovy |
| Platform | Server |

### temurin-build

| | |
|---|---|
| Repository | https://github.com/adoptium/temurin-build |
| Version | `da2408e4ea988090835f15f29cb170873cced045` |
| Type | Bash |
| Platform | Server |

## Project Coverage

This section provides an overview of the analysis coverage of the review, as determined by our high-level engagement goals. Our approaches included the following:

- Static analysis of the codebase using Semgrep rules, focused on the following:
    - Connections to HTTP endpoints
    - Connections to HTTPS endpoints
    - Software authenticity verification
    - Hostname or host key verification
    - Excessive user privileges
    - Basic Java and Kotlin code quality issues
- Manual review of the codebase, focused on the following:
    - Authentication, authorization, and access controls
    - Command injection and other forms of injection bugs
    - SSL hostname verification, authenticity, and integrity validation
- Binary analysis focused on various forms of security hardening using the `checksec` tool

### Coverage Limitations

Because of the time-boxed nature of testing work, it is common to encounter coverage limitations. The following list outlines the coverage limitations of the engagement and indicates system elements that may warrant further review:

- We did not review each of Temurin's dependencies to ensure that they are up to date and secure.
    - We did not review the `temurin-build/security/mk-ca-bundle.pl` file, which was made by the cURL project.

# Automated Testing

Trail of Bits uses automated techniques to extensively test the security properties of software. We use both open-source static analysis and fuzzing utilities, along with tools developed in house, to perform automated testing of source code and compiled software.

## Test Harness Configuration

We used the following tools in the automated testing phase of this project:

| Tool | Description | Policy |
|---|---|---|
| Semgrep | An open-source static analysis tool for finding bugs and enforcing code standards when editing or committing code and during build time | Rules to be provided in accompanying zip file |
| `checksec` | An open-source binary analysis tool for checking security properties of executables like PIE, RELRO, stack canaries, ASLR, and source fortification | Default |
| `route-detect` | An open-source static analysis tool for finding authentication and authorization security bugs in web application routes | Default, with Java Jakarta package namespace added |

## Areas of Focus

Our automated testing and verification work focused on the following system properties:

- Secure HTTP downloads and endpoint access
- Authenticity and integrity guarantees
- Least privilege access controls
- Binary hardening flags
- Web application route authentication and authorization controls

Our testing work focused on finding the following types of issues:

- Cryptographic weaknesses and insecure algorithms
- Hard-coded or exposed secrets, credentials, and tokens

## Codebase Maturity Evaluation

Trail of Bits uses a traffic-light protocol to provide each client with a clear understanding of the areas in which its codebase is mature, immature, or underdeveloped. Deficiencies identified here often stem from root causes within the software development life cycle that should be addressed through standardization measures (e.g., the use of common libraries, functions, or frameworks) or training and awareness programs.

| Category | Summary | Result |
|---|---|---|
| Arithmetic | Due to the nature of the project, Temurin makes minimal use of arithmetic. | **Not Applicable** |
| Auditing | Log messages are saved while building and testing OpenJDK releases. The API server tracks telemetry data using Microsoft Application Insights. Both of these features will make incident response much easier in the event of a security problem. | **Strong** |
| Authentication / Access Controls | End-user authentication is minimal and is soundly implemented where necessary. Access controls are generally limited to least privilege, with the exception of the controls described in TOB-TEMURIN-18. Although lower priority, improvements to password management can be made to address TOB-TEMURIN-8, TOB-TEMURIN-10, and TOB-TEMURIN-11. | **Satisfactory** |
| Complexity Management | Code complexity varies across repositories. For example, the `api.adoptium.net` repository is well structured, while the `temurin-build` repository lacks inherent structure. The use of a modern programming language for the API server versus the ad hoc scripting in the build repository may account for this discrepancy. In many cases, good documentation accompanies disorganized code, making it easier to manage and understand. | **Moderate** |
| Configuration | The targets rely heavily on Ansible for configuration management. The `infrastructure` codebase contains a significant number of insecure configurations, such as those described in TOB-TEMURIN-4, TOB-TEMURIN-5, and TOB-TEMURIN-13, and multiple code quality issues, described in appendix E. Configuration practices related | **Weak** |

| | | |
|---|---|---|
| | to password and dependency management can also be improved. | |
| Cryptography and Key Management | We found multiple places where important cryptography features, such as HTTPS downloads and signature verifications, are disabled. A large portion of the findings in this report are related to these disabled cryptography features (TOB-TEMURIN-3, TOB-TEMURIN-4, TOB-TEMURIN-5, TOB-TEMURIN-7, TOB-TEMURIN-9, TOB-TEMURIN-12, TOB-TEMURIN-13, TOB-TEMURIN-16). | **Weak** |
| Data Handling | In general, we found that Temurin correctly handles its data (such as its binaries). However, we noticed multiple cases in which command injection is possible (TOB-TEMURIN-1, TOB-TEMURIN-17). We also noticed one case in which a Red Hat password is exposed (TOB-TEMURIN-8). In addition, issues related to cryptography (see above) often have the potential to lead to leakage or corruption of Temurin's data. | **Satisfactory** |
| Documentation | Temurin provides comprehensive documentation describing the project layout and build process. READMEs, network diagrams, a basic threat model, and other thorough, text-based documentation describe necessary workflows and design decisions. Code is commented where needed. | **Strong** |
| Maintenance | Dependabot is used to keep the `api.adoptium.net`, `ci-jenkins-pipelines`, and `temurin-build` repositories up to date. Dependabot was incorrectly added in the `infrastructure` repository (TOB-TEMURIN-19). (It is not used in the `jenkins-helper` repository since the repository does not specify any dependencies.) In most cases, dependencies are pinned using a checksum or verified using a signature; however, there are a number of exceptions to this (TOB-TEMURIN-3, TOB-TEMURIN-4, TOB-TEMURIN-5, TOB-TEMURIN-6, TOB-TEMURIN-9, TOB-TEMURIN-12, TOB-TEMURIN-15, TOB-TEMURIN-16). Additionally, in the `temurin-build` repository, dependencies are downloaded in many different places in an ad hoc fashion, making it difficult to determine the full list of dependencies being used. We recommend, at the very least, documenting a list of all locations where | **Weak** |

| | | |
|---|---|---|
| | these downloads are performed (see the executive summary). | |
| Memory Safety and Error Handling | The Temurin project uses memory-safe languages, with very few exceptions. We did not find any issues related to memory safety or error handling. | **Strong** |
| Testing and Verification | Tests are run on JDK builds as part of the pipeline to ensure correctness. The `api.adoptium.net` repository includes tests for its Kotlin code, including tests for both happy-path and unhappy-path behavior. Temurin would benefit from having Semgrep run on each new PR submitted to each of its repositories. | **Satisfactory** |

## Summary of Findings

The table below summarizes the findings of the review, including type and severity details.

| ID | Title | Type | Severity |
|---|---|---|---|
| 1 | Command injection vulnerability in WinRM script | Data Validation | High |
| 2 | Docker Compose ports exposed on all interfaces | Configuration | Low |
| 3 | Insecure installation of Xcode software | Cryptography | High |
| 4 | Insecure software downloads in Ansible playbooks | Cryptography | High |
| 5 | Signature verification disabled during software installation | Cryptography | High |
| 6 | Missing integrity check in Dragonwell Dockerfile | Cryptography | Low |
| 7 | Hostname verification disabled on MongoDB client | Cryptography | High |
| 8 | RHEL build image includes password | Data Exposure | Low |
| 9 | Insecure downloads using wget command | Cryptography | High |
| 10 | Hard-coded CA bundle keystore password | Access Controls | Informational |
| 11 | Hard-coded Vagrant VM password | Access Controls | Informational |
| 12 | Missing integrity or authenticity check in jcov script download | Cryptography | Low |
| 13 | SSH client disables host key verification | Cryptography | High |
| 14 | Compiler mitigations are not enabled | Configuration | Informational |
| 15 | Use of unpinned third-party workflows | Patching | Medium |

| | | | |
|---|---|---|---|
| 16 | Third-party dependencies used without signature or checksum verification | Patching | **Informational** |
| 17 | Code injection vulnerability in build-scripts pipeline jobs | Data Validation | **High** |
| 18 | Docker commands specify root user in containers | Access Controls | **Informational** |
| 19 | Incorrect Dependabot configuration filename | Patching | **Undetermined** |

## Detailed Findings

| 1. Command injection vulnerability in WinRM script | |
|---|---|
| Severity: **High** | Difficulty: **High** |
| Type: Data Validation | Finding ID: TOB-TEMURIN-1 |
| Target: `infrastructure/ansible/pbTestScripts/startScriptWin.py` | |

### Description

The `run_winrm` function allows callers to specify commands to run on a Vagrant virtual machine (VM). It is the primary functionality of the `startScriptWin.py` script, which is itself executed by the `vagrantPlaybookCheck.sh` shell script. This function receives input from command-line arguments and uses string concatenation to build a shell command to execute on a Vagrant VM:

```
def run_winrm(vmIP, buildArgs, mode):
    cmd_str = "Start-Process powershell.exe -Verb runAs; cd C:/tmp; sh 
C:/vagrant/pbTestScripts/"
    print(mode)
    if mode == 1:
        cmd_str += "buildJDKWin.sh "
    else:
        cmd_str += "testJDKWin.sh "
    cmd_str += buildArgs
    print("Running :      %s" %cmd_str)
    session = winrm.Session(str(vmIP), auth=('vagrant', 'vagrant'))
    session.run_ps(cmd_str, sys.stdout, sys.stderr)
```

*Figure 1.1: A shell command generated with string concatenation (`infrastructure/ansible/pbTestScripts/startScriptWin.py:12–22`)*

If an attacker can influence the `buildArgs` parameter, either through the `startScriptWin.py` or `vagrantPlaybookCheck.sh` command-line arguments, then they could be able to execute code on the Vagrant VM. The Eclipse Foundation has confirmed that these parameters can be specified in a Jenkins job web form; however, access to these forms is restricted.

### Exploit Scenario

An attacker sends a malicious shell payload through the `--build-fork` or `--build-branch` command-line argument to `vagrantPlaybookCheck.sh`, or through the `-a` command-line argument to `startScriptWin.py`. While building the `cmd_str`, the run_winrm function concatenates the buildArgs string and executes it on the Vagrant VM. The attacker is able to execute arbitrary commands by using shell operators such as ;, &&, or ||, and to append additional commands.

It is worth noting that spaces cannot be used in the payload if it is sent to vagrantPlaybookCheck.sh. However, shell brace expansion can be used to bypass this restriction. For example, the following command results in successful command injection:

```
./vagrantPlaybookCheck.sh ... --branch main;{echo,command,injection}; ...
```

**Recommendations**

Short term, build a list of command arguments to be passed to the run_winrm method instead of using string concatenation to generate a command argument string and passing it to run_ps.

Long term, implement static analysis rules to automatically detect string concatenation data that is passed to the run_ps method.

## 2. Docker Compose ports exposed on all interfaces

| | |
|---|---|
| Severity: **Low** | Difficulty: **High** |
| Type: Configuration | Finding ID: TOB-TEMURIN-2 |
| Target: `api.adoptium.net/docker-compose.yml` | |

### Description

The `docker-compose.yml` configuration file for the `api.adoptium.net` API server (which is used in development but not in production) specifies Docker ports using a `ports` configuration option of `27017:27017` for the MongoDB container and `8080:8080` for the front-end container (see figure 2.1). This means that these ports are accessible not just to other processes running on the same computer, but also from other computers on the same network.

```
version: '3.6'
services:
  mongodb:
    image: mongo:4.2
    ports:
      - "27017:27017"
  frontend:
    depends_on:
      - mongodb
    image: "adoptium-api"
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - "8080:8080"
    environment:
      MONGODB_HOST: mongodb
  updater:
    depends_on:
      - mongodb
    image: "adoptium-api"
    command: "java -jar /deployments/adoptium-api-v3-updater-runner.jar"
    build:
      context: .
      dockerfile: Dockerfile
    environment:
      MONGODB_HOST: mongodb
      GITHUB_TOKEN: "${GITHUB_TOKEN}"
      GITHUB_APP_ID: "${GITHUB_APP_ID}"
      GITHUB_APP_PRIVATE_KEY: "${GITHUB_APP_PRIVATE_KEY}"
    GITHUB_APP_INSTALLATION_ID: "${GITHUB_APP_INSTALLATION_ID}"
```

*Figure 2.1:* `api.adoptium.net/docker-compose.yml`

### Exploit Scenario

A Temurin developer runs this `docker-compose.yml` file while on a public Wi-Fi network. An attacker who is on the same network connects to the MongoDB database running on the developer's computer; this database is available on port `27017` without any password protection. The attacker modifies an entry in the database containing a link to a binary file, which eventually causes the developer to unwittingly download and run a malicious file.

### Recommendations

Short term, set these configuration values to `127.0.0.1:27017:27017` and `127.0.0.1:8080:8080`, instead of `27017:27017` and `8080:8080`.

Long term, use static analysis rules to automatically detect ports that are exposed on all interfaces; the set of Semgrep rules provided alongside this report includes such a rule.

| 3. Insecure installation of Xcode software | |
|---|---|
| Severity: **High** | Difficulty: **High** |
| Type: Cryptography | Finding ID: TOB-TEMURIN-3 |
| Target: `infrastructure/ansible/playbooks/AdoptOpenJDK_ITW_Playbook/roles/Common/scripts/install-xcode.sh` | |

## Description

The `install-xcode.sh` script uses unencrypted HTTP endpoints to download Xcode command-line tools and then installs them using the `-allowUntrusted` flag:

```
if [[ "$osx_vers" -eq 7 ]] || [[ "$osx_vers" -eq 8 ]]; then

        if [[ "$osx_vers" -eq 7 ]]; then
DMGURL=http://devimages.apple.com/downloads/xcode/command_line_tools_for_xcode_os_x_lion_april_
2013.dmg
        fi

        if [[ "$osx_vers" -eq 8 ]]; then
DMGURL=http://devimages.apple.com/downloads/xcode/command_line_tools_for_osx_mountain_lion_apri
l_2014.dmg
        fi

                TOOLS=cltools.dmg
                curl "$DMGURL" -o "$TOOLS"
                TMPMOUNT=`/usr/bin/mktemp -d /tmp/clitools.XXXX`
                hdiutil attach "$TOOLS" -mountpoint "$TMPMOUNT" -nobrowse
                # The "-allowUntrusted" flag has been added to the installer
                # command to accomodate for now-expired certificates used
                # to sign the downloaded command line tools.
                installer -allowUntrusted -pkg "$(find $TMPMOUNT -name '*.mpkg')" -target /
                hdiutil detach "$TMPMOUNT"
                rm -rf "$TMPMOUNT"
                rm "$TOOLS"
fi
```

*Figure 3.1: Untrusted installation of Xcode software (`infrastructure/ansible/playbooks/AdoptOpenJDK_ITW_Playbook/roles/Common/scripts/install-xcode.sh:23–44`)*

Also, the OS X version check performs an imprecise comparison. This increases the likelihood that the untrusted installation will be performed on versions it is not intended for. The `osx_vers` variable considers only the system minor version rather than the minor and major version:

```
osx_vers=$(sw_vers -productVersion | awk -F "." '{print $2}')
```

*Figure 3.2: The code checks only the system minor version.*
(`infrastructure/ansible/playbooks/AdoptOpenJDK_ITW_Playbook/roles/Common/scripts/install-xcode.sh:2`)

This script is meant to perform the untrusted installation only if it is running on OS X version 10.7 or 10.8. Because the code checks only the minor version, this script will also perform the untrusted installation on macOS versions 11.7, 12.7, 13.7, and so on.

### Exploit Scenario

An attacker is in a privileged network position relative to a system installing Xcode software and is able to actively intercept and modify the system's network traffic. Because the software is downloaded over HTTP and its installation is untrusted, the attacker can modify the download in transit and replace the software with a malicious version.

### Recommendations

Short term, have the script use HTTPS to download the software and ensure the integrity of the software by validating it against a known SHA-256 checksum.

Long term, deprecate and remove support for OS X and macOS versions requiring an untrusted installation of the Xcode command-line tools.

| 4. Insecure software downloads in Ansible playbooks | |
|---|---|
| Severity: **High** | Difficulty: **High** |
| Type: Cryptography | Finding ID: TOB-TEMURIN-4 |
| Target: The full list of targets is provided in appendix C. | |

**Description**

Ansible playbooks are used to configure various parts of the system infrastructure. These playbooks install software and generally configure systems to be in a consistent state. Many of the playbooks install software and package data in an insecure manner, using unencrypted channels such as HTTP (figure 4.1) or disabling certificate validation when performing the download (figure 4.2). The full list of such instances is provided in appendix C.

```
- name: Add Azul Zulu GPG Package Signing Key for x86_64
  apt_key:
    url: http://repos.azulsystems.com/RPM-GPG-KEY-azulsystems
    state: present
  when:
    - ansible_architecture == "x86_64"
  tags: [patch_update, azul-key]
```

*Figure 4.1: HTTP download (`infrastructure/ansible/playbooks/AdoptOpenJDK_Unix_Playbook/roles/Common/tasks/Ubuntu.yml:25–31`)*

```
- name: Enable EPEL release (not CentOS8)
  yum:
    name: epel-release
    state: installed
    update_cache: yes
    validate_certs: no
  when: ansible_distribution_major_version != "8"
  tags: patch_update
```

*Figure 4.2: Disabled SSL certificate validation (`infrastructure/ansible/playbooks/AdoptOpenJDK_Unix_Playbook/roles/Common/tasks/CentOS.yml:15–22`)*

Note that there are many more instances in which `validate_certs` is disabled. However, packages or downloads that specify a checksum alongside disabled validation are considered secure. This configuration was assumed to mean "trust on first use" and that integrity of the software has been verified out of band and validated with a checksum. An example of the configuration is provided below:

```
- name: Download expat
  get_url:
    url:
https://github.com/libexpat/libexpat/releases/download/R_2_2_5/expat-2.2.5.tar.bz2
    dest: /tmp/
    mode: 0440
    timeout: 25
    validate_certs: no
    checksum:
sha256:d9dc32efba7e74f788fcc4f212a43216fc37cf5f23f4c2339664d473353aedf6
```

*Figure 4.3: SSL certificate validation disabled and checksum provided (`infrastructure/ansible/playbooks/AdoptOpenJDK_Unix_Playbook/roles/Common/tasks/openSUSE.yml:151–158`)*

## Exploit Scenario

An attacker is in a privileged network position relative to a system installing software using an Ansible playbook and is able to actively intercept and modify the system's network traffic. Because the software is downloaded over HTTP, the attacker can modify the download in transit and replace the software with a malicious version.

## Recommendations

Short term, change HTTP downloads to HTTPS, and enable SSL certificate validation.

Long term, implement static analysis rules to automatically detect HTTP downloads and disabled SSL certificate validation in Ansible playbooks.

## 5. Signature verification disabled during software installation

| | |
|---|---|
| Severity: **High** | Difficulty: **High** |
| Type: Cryptography | Finding ID: TOB-TEMURIN-5 |
| Target: `infrastructure/ansible/playbooks/AdoptOpenJDK_Unix_Playbook/roles/Common/tasks/openSUSE.yml`, `infrastructure/ansible/playbooks/AdoptOpenJDK_Unix_Playbook/roles/NVidia_Cuda_Toolkit/tasks/main.yml`, `infrastructure/ansible/playbooks/AdoptOpenJDK_Unix_Playbook/roles/Docker/tasks/rhel.yml` | |

### Description

Software package signatures are verified upon installation to ensure their authenticity. GNU Privacy Guard (GPG) signatures are a common signing method. A number of Ansible playbooks disable GPG verification when installing packages. The following snippets show four locations where verification is disabled:

```
- name: Sed change gpgcheck for gcc repo on x86_64
  replace:
    path: /etc/zypp/repos.d/devel_gcc.repo
    regexp: 'gpgcheck=1'
    replace: "gpgcheck=0"
  when:
    - (ansible_distribution_major_version == "12" and ansible_architecture ==
"x86_64")
  tags: SUSE_gcc48
```

*Figure 5.1: openSUSE playbook disabling GPG verification (`infrastructure/ansible/playbooks/AdoptOpenJDK_Unix_Playbook/roles/Common/tasks/openSUSE.yml:29–36`)*

```
- name: Sed change gpgcheck for SLES12 on x86_64
  command: sed 's/gpgcheck=1/gpgcheck=0/' -i /etc/zypp/repos.d/cuda.repo
  when:
    - cuda_installed.stat.islnk is not defined
    - ansible_architecture == "x86_64"
    - ansible_distribution == "SLES" or ansible_distribution == "openSUSE"
    - ansible_distribution_major_version == "12"
  tags:
    - nvidia_cuda_toolkit
    #TODO: rpm used in place of yum or rpm_key module
    - skip_ansible_lint
```

*Figure 5.2: NVIDIA playbook disabling GPG verification (`infrastructure/ansible/playbooks/AdoptOpenJDK_Unix_Playbook/roles/NVidia_Cuda_Toolkit/tasks/main.yml:105–115`)*

```
- name: Add Docker Repo x86-64/ppc64le
  yum_repository:
    name: docker
    description: docker repository
    baseurl: "https://download.docker.com/linux/centos/{{
ansible_distribution_major_version }}/{{ ansible_architecture }}/stable"
    enabled: true
    gpgcheck: false
  when:
    - ansible_architecture == "x86_64" or ansible_architecture == "ppc64le"

- name: Add Docker repo for s390x on RHEL
  yum_repository:
    name: docker
    description: docker YUM repo s390x
    baseurl: https://download.docker.com/linux/rhel/{{
ansible_distribution_major_version }}/s390x/stable/
    enabled: true
    gpgcheck: false
  when:
    - ansible_architecture == "s390x"
```

*Figure 5.3: Docker playbook disabling GPG verification (`infrastructure/ansible/playbooks/AdoptOpenJDK_Unix_Playbook/roles/Docker/tasks/rhel.yml:13–31`)*

## Exploit Scenario

An attacker wants to upload a malicious package to one of the repositories. He is able to bypass the repository signing process or sign the package with an untrusted GPG key and successfully upload the package. The system performing the installation then installs the malicious package despite receiving an incorrect signature, or no signature at all.

## Recommendations

Short term, import the correct package repository GPG keys, and enable GPG signature verification.

Long term, implement static analysis rules to automatically detect disabled GPG signature verification in Ansible playbooks.

## 6. Missing integrity check in Dragonwell Dockerfile

| | |
|---|---|
| Severity: **Low** | Difficulty: **High** |
| Type: Cryptography | Finding ID: TOB-TEMURIN-6 |
| Target: `ci-jenkins-pipelines/pipelines/build/dockerFiles/dragonwell.dockerfile` | |

### Description

The Dragonwell Dockerfile downloads and installs the Dragonwell software without verifying its integrity. A hashsum like SHA-256 should be used to ensure the integrity of the download and that the system is receiving the same data across multiple downloads.

```
RUN \
    # Dragonewell 8 requires a dragonwell 8 BootJDK
    mkdir -p /opt/dragonwell; \
    wget
https://github.com/alibaba/dragonwell8/releases/download/dragonwell-8.4.4_jdk8u262-g
a/Alibaba_Dragonwell_8.4.4-GA_Linux_x64.tar.gz; \
    tar -xf Alibaba_Dragonwell_8.4.4-GA_Linux_x64.tar.gz -C /opt/; \
    mv /opt/jdk8u262-b10 /opt/dragonwell8
```

*Figure 6.1: Download of the Dragonwell software* (`ci-jenkins-pipelines/pipelines/build/dockerFiles/dragonwell.dockerfile:5–10`)

Note that the equivalent AArch64 download of the same software does verify the integrity with an MD5 hashsum:

```
RUN \
    # Dragonewell 8 requires a dragonwell 8 BootJDK
    mkdir -p /opt/dragonwell8; \
    wget
https://github.com/alibaba/dragonwell8/releases/download/dragonwell-8.5.5_jdk8u275-b
2/Alibaba_Dragonwell_8.5.5-FP1_Linux_aarch64.tar.gz; \
    test $(md5sum Alibaba_Dragonwell_8.5.5-FP1_Linux_aarch64.tar.gz | cut -d ' '
-f1) = "ab80c4f638510de8c7211b7b7734f946" || exit 1; \
    tar -xf Alibaba_Dragonwell_8.5.5-FP1_Linux_aarch64.tar.gz -C /opt/dragonwell8
--strip-components=1
```

*Figure 6.2: Download of the AArch64 Dragonwell software* (`ci-jenkins-pipelines/pipelines/build/dockerFiles/dragonwell_aarch64.dockerfile:5–10`)

### Exploit Scenario
An attacker is able to upload a malicious package to one of the repositories. The system performing the installation then installs the malicious package even though the underlying data within the package has changed.

### Recommendations
Short term, add a SHA-256 hashsum check to ensure the integrity of the software.

## 7. Hostname verification disabled on MongoDB client

| | |
|---|---|
| Severity: **High** | Difficulty: **High** |
| Type: Cryptography | Finding ID: TOB-TEMURIN-7 |
| Target: `api.adoptium.net/adoptium-api-v3-persistence/src/main/kotlin/net/adoptium/api/v3/dataSources/persitence/mongo/MongoClient.kt` | |

### Description

The MongoDB client used by the API server disables hostname verification when SSL is enabled. This could enable attackers to steal the database username and password through person-in-the-middle attacks.

```
var settingsBuilder = MongoClientSettings.builder()
    .applyConnectionString(ConnectionString(connectionString))
val sslEnabled = System.getenv("MONGODB_SSL")?.toBoolean()
if (sslEnabled == true) {
    settingsBuilder = settingsBuilder.applyToSslSettings {
it.enabled(true).invalidHostNameAllowed(true) }
}
client = KMongo.createClient(settingsBuilder.build()).coroutine
database = client.getDatabase(dbName)
```

*Figure 7.1: Configuration code that disables hostname verification (`api.adoptium.net/adoptium-api-v3-persistence/src/main/kotlin/net/adoptium/api/v3/dataSources/persitence/mongo/MongoClient.kt#67–74`)*

### Exploit Scenario

The API server sends a request to the MongoDB database. A person-in-the-middle attacker impersonates the database, using his own SSL key. The API server then sends over its database username and password, encrypted using the attacker's public key, rather than the database's public key. The attacker now knows the database's username and password and can tamper with its contents.

### Recommendations

Short term, enable hostname verification by removing the call to `invalidHostNameAllowed`.

|  |  |
|---|---|
| **8. RHEL build image includes password** | |
| Severity: **Low** | Difficulty: **High** |
| Type: Data Exposure | Finding ID: TOB-TEMURIN-8 |
| Target: `infrastructure/ansible/docker/Dockerfile.RHEL7` | |

## Description

The Red Hat Enterprise Linux (RHEL) build image takes a Red Hat username and password as a build argument. Docker build arguments are persisted in the resulting image, meaning that anyone who gains access to Temurin's RHEL image will also have access to the Red Hat login information.

```
FROM registry.access.redhat.com/rhel7
# This dockerfile should be built using:
#  docker build --no-cache -t rhel7_build_image -f ansible/docker/Dockerfile.RHEL7
--build-arg ROSIUSER=******* --build-arg ROSIPW=******* --build-arg git_sha=*******
`pwd`
ARG ROSIUSER
ARG ROSIPW
RUN sed -i 's/\(def in_container():\)/\1\n    return False/g'
/usr/lib64/python*/*-packages/rhsm/config.py
RUN subscription-manager register --username=${ROSIUSER} --password=${ROSIPW}
--auto-attach
```

*Figure 8.1:* `infrastructure/ansible/docker/Dockerfile.RHEL7#1–7`

## Recommendations

Short term, use build secrets, rather than build arguments, to provide login information.

## 9. Insecure downloads using wget command

| | |
|---|---|
| Severity: **High** | Difficulty: **High** |
| Type: Cryptography | Finding ID: TOB-TEMURIN-9 |
| Target: `jenkins-helper/Jenkins_jobs/CreateNewNode.groovy`, `infrastructure/ansible/playbooks/AdoptOpenJDK_Unix_Playbook/roles/DockerStatic/Dockerfiles/Dockerfile`, `infrastructure/docs/Setup-QEMU-Images.md` | |

### Description

The `wget` command is used to download data over a network. The target codebases use `wget` in an insecure manner in a number of locations, using unencrypted channels such as HTTP or disabling certificate validation when performing the download. The following snippets show five locations where `wget` is used in an insecure manner:

```
RUN wget
'http://mirror.centos.org/centos/8-stream/BaseOS/x86_64/os/Packages/centos-gpg-keys-
8-3.el8.noarch.rpm' -O /tmp/gpgkey.rpm
RUN rpm -i '/tmp/gpgkey.rpm'
RUN wget
'http://mirror.centos.org/centos/8-stream/BaseOS/x86_64/os/Packages/centos-stream-re
pos-8-3.el8.noarch.rpm' -O /tmp/centosrepos.rpm
```

*Figure 9.1: Unencrypted, HTTP download (`infrastructure/ansible/playbooks/AdoptOpenJDK_Unix_Playbook/roles/DockerStatic/Dockerfiles/Dockerfile.ubi8:7–9`)*

```
wget -O installer-vmlinuz
http://http.us.debian.org/debian/dists/jessie/main/installer-armhf/current/images/ne
tboot/vmlinuz
wget -O installer-initrd.gz
http://http.us.debian.org/debian/dists/jessie/main/installer-armhf/current/images/ne
tboot/initrd.gz
```

*Figure 9.2: Unencrypted, HTTP download (`infrastructure/docs/Setup-QEMU-Images.md:166–167`)*

```
launcher = new CommandLauncher(Constants.SSH_COMMAND + "${machineIPs[index]} " +
"\"wget -q --no-check-certificate -O slave.jar ${JENKINS_URL}jnlpJars/slave.jar ;
java -jar slave.jar\"");
```

*Figure 9.3: Download with certificate validation disabled (`jenkins-helper/Jenkins_jobs/CreateNewNode.groovy:32`)*

## Exploit Scenario

An attacker is in a privileged network position relative to a system downloading data using `wget` and is able to actively intercept and modify the system's network traffic. Because the data is downloaded without SSL certificate verification, the attacker can modify the download in transit and replace the data with a malicious version.

## Recommendations

Short term, change HTTP downloads to HTTPS, and enable SSL certificate validation. If it is not possible to change an HTTP download to HTTPS, such as in a package installation, then a verification key such as a GPG key should be included out of band and used to verify the package installation. In other words, a key can be hard-coded into an installation procedure and used to "trust on first use."

Long term, implement static analysis rules to automatically detect HTTP downloads and disabled SSL certificate validation in `wget` commands.

## 10. Hard-coded CA bundle keystore password

| Severity: **Informational** | Difficulty: **High** |
|---|---|
| Type: Access Controls | Finding ID: TOB-TEMURIN-10 |
| Target: `api.adoptium.net/deploy/run.sh`, `temurin-build/security/mk-cacerts.sh` | |

### Description

The password used for the certificate authority (CA) bundle generated for the API service is hard-coded as `changeit`:

```
keytool -import -alias mongodb -storepass changeit -keystore ./cacerts -file
"${MONGO_CERT_FILE}" -noprompt
JAVA_OPTS="$JAVA_OPTS -Djavax.net.ssl.trustStore=./cacerts
-Djavax.net.ssl.trustStorePassword=changeit"
```

*Figure 10.1: Hard-coded password (`api.adoptium.net/deploy/run.sh:29–30`)*

```
echo "Processing certificate with alias: $ALIAS"
"$KEYTOOL" -noprompt \
-import \
-storetype JKS \
-alias "$ALIAS" \
-file "$FILE" \
-keystore "cacerts" \
-storepass "changeit"
...
num_certs=$("$KEYTOOL" -v -list -storepass changeit -keystore cacerts | grep -c
"Alias name:")
```

*Figure 10.2: Hard-coded password (`temurin-build/security/mk-cacerts.sh:118–125, 143`)*

This CA bundle is generated in a deterministic manner from publicly available Mozilla certificate data. This may seem to indicate that it need not be password-protected. However, the keystore password is used to verify the integrity and authenticity of the bundle. Without a confidential password set, the integrity and authenticity of the data cannot be verified as the data moves from the build to runtime environment. Due to the lack of potentially attacker-controlled inputs into this functionality, this finding's severity is set to informational.

### Recommendations

Short term, use a strong, randomly generated password to store this keystore data, and include this password at runtime to verify the authenticity of the CA bundle data.

## 11. Hard-coded Vagrant VM password

| | |
|---|---|
| Severity: **Informational** | Difficulty: **High** |
| Type: Access Controls | Finding ID: TOB-TEMURIN-11 |
| Target: `infrastructure/ansible/pbTestScripts/startScriptWin.py` | |

### Description

Vagrant VMs are used to execute build and test workloads in a CI environment. The VMs use a hard-coded password for authentication:

```
session = winrm.Session(str(vmIP), auth=('vagrant', 'vagrant'))
session.run_ps(cmd_str, sys.stdout, sys.stderr)
```

*Figure 11.1: Hard-coded password (`infrastructure/ansible/pbTestScripts/startScriptWin.py:21–22`)*

These VMs are run on an internal system without public access and are discarded upon completion of the workload. Due to the ephemeral nature of these VMs, the severity of this finding is set to informational. However, using a strong, random password may limit lateral movement in the event of an unrelated compromise and would be a beneficial defense-in-depth mechanism.

### Recommendations

Short term, use a strong, randomly generated password to authenticate Vagrant VMs at runtime.

## 12. Missing integrity or authenticity check in jcov script download

| | |
|---|---|
| Severity: **Low** | Difficulty: **High** |
| Type: Cryptography | Finding ID: TOB-TEMURIN-12 |
| Target: `ci-jenkins-pipelines/tools/code-tools/jcov.sh` | |

### Description

The `jcov.sh` script downloads ASM tools without verifying their integrity or authenticity:

```
local tools="asm asm-tree asm-util"
local main_url="https://repository.ow2.org/nexus/content/repositories/releases/org/ow2/asm"
ASM_TITLE="Built against '$tools' tools in version '$asm_version'"
ASM_URLS=""
ASM_JARS=""
ASM_PROPS=""
for tool in $tools; do
  local tool_prop="`echo $tool|sed "s/-/./g"`.jar"
  local tool_versioned="$tool-$asm_version.jar"
  local tool_url="$main_url/$tool/$asm_version/$tool_versioned"
  if [ "$asm_manual" == "true" ] ; then
    if [ ! -e $tool_versioned ] ; then
      wget $tool_url
    fi
    ...
```

*Figure 12.1: Download missing integrity or authenticity check*
*(`ci-jenkins-pipelines/tools/code-tools/jcov.sh:65–78`)*

The integrity or authenticity should be verified using a hashsum like SHA-256 or a signature like a GPG signature. This would ensure that the system is receiving the same data across multiple downloads. This download does use HTTPS, so this issue is marked as low severity.

### Exploit Scenario

An attacker is able to upload a malicious package to one of the repositories. The system performing the installation then installs the malicious package even though the underlying data within the package has changed.

### Recommendations

Short term, add a SHA-256 hashsum check to ensure the integrity of the software, or a GPG verification to ensure the authenticity of the software. Both mechanisms are made available by the `repository.ow2.org` ASM repository.

## 13. SSH client disables host key verification

| | |
|---|---|
| Severity: **High** | Difficulty: **High** |
| Type: Cryptography | Finding ID: TOB-TEMURIN-13 |
| Target: `infrastructure/ansible/playbooks/Supporting_Scripts/Nagios_Ansible_Config_tool/Nagios_RemoteTunnel.sh`, `infrastructure/ansible/playbooks/Supporting_Scripts/Nagios_Ansible_Config_tool/Nagios_Ansible_Config_tool.sh`, `infrastructure/ansible/playbooks/AdoptOpenJDK_Unix_Playbook/roles/Nagios_Master_Config/tasks/main.yml` | |

### Description

SSH clients maintain a list of known-good hosts they have connected to before. Host key verification is then used to prevent person-in-the-middle attacks. There are a number of locations across the target repositories that disable SSH host key verification, such as when connecting to a Nagios instance:

```
Reverse_Tunnel="ssh -o StrictHostKeyChecking=no -f -n -N -R $REMOTE_PORT:127.0.0.1:$LOCAL_PORT
$USER_NAME@$REMOTE_HOST -p $LOGIN_PORT -i $IDENTITY_KEY"
```

*Figure 13.1: Nagios SSH connection disabling SSH host key verification (`infrastructure/ansible/playbooks/Supporting_Scripts/Nagios_Ansible_Config_tool/Nagios_RemoteTunnel.sh:18–21`)*

```
Nagios_Login=`su nagios -c "ssh -o StrictHostKeyChecking=no $Sys_IPAddress uptime"`
```

*Figure 13.2: Nagios SSH connection disabling SSH host key verification (`infrastructure/ansible/playbooks/Supporting_Scripts/Nagios_Ansible_Config_tool/Nagios_Ansible_Config_tool.sh:170`)*

```
command: ssh -o StrictHostKeyChecking=no root@{{ Nagios_Master_IP }}
"/usr/local/nagios/Nagios_Ansible_Config_tool/Nagios_Ansible_Config_tool.sh  {{
ansible_distribution }} {{ ansible_architecture }} {{ inventory_hostname }} {{
ansible_host }} {{ provider }} {{ ansible_port }} "
```

*Figure 13.3: Nagios SSH connection disabling SSH host key verification (`infrastructure/ansible/playbooks/AdoptOpenJDK_Unix_Playbook/roles/Nagios_Master_Config/tasks/main.yml:25`)*

There are also a number of benign locations where SSH host key verification is disabled. These locations are considered benign because they are connecting to internal, short-lived, or local-only services. They are included here for completeness's sake:

```
launcher = new SSHLauncher(
            machines[index],
            22,
            params.SSHCredentialId.isEmpty() ? Constants.SSH_CREDENTIAL_ID :
params.SSHCredentialId,
            null, null, null, null, null, null, null,
            new NonVerifyingKeyVerificationStrategy());
```

*Figure 13.4: Groovy SSH launcher disabling host key verification (`jenkins-helper/Jenkins_jobs/CreateNewNode.groovy:38–43`)*

```
sshpass -p 'password' ssh linux@localhost -p "$PORTNO" -o StrictHostKeyChecking=no 'uname -a'
```

*Figure 13.5: Test script disabling host key verification (`infrastructure/ansible/pbTestScripts/qemuPlaybookCheck.sh:273`)*

```
ssh_args="$ssh_args -o StrictHostKeyChecking=no"
```

*Figure 13.6: Test script disabling host key verification (`infrastructure/ansible/pbTestScripts/vagrantPlaybookCheck.sh:253`)*

### Exploit Scenario

An attacker is in a privileged network position relative to a system initiating an SSH connection and is able to actively intercept and modify the system's network traffic. Because SSH host key verification is disabled, the attacker can intercept SSH network traffic and perform a person-in-the-middle attack.

### Recommendations

Short term, in all locations where SSH host key verification is currently disabled, have the code gather the host's SSH public key and add it out of band to the client's `known_hosts` file.

Long term, implement static analysis rules to automatically detect when SSH host key verification is disabled.

## 14. Compiler mitigations are not enabled

| | |
|---|---|
| Severity: **Informational** | Difficulty: **High** |
| Type: Configuration | Finding ID: TOB-TEMURIN-14 |
| Target: `temurin-build/sbin/build.sh` | |

### Description

The Temurin build does not have all modern compiler security mitigations enabled. This makes it easier for an attacker who finds a low-level vulnerability to exploit it and gain control over the process. Modern compilers support exploit mitigations such as the following:

- **Non-executable flag:** Marks the program's data sections as non-executable
- **PIE flag:** Makes the program compiled as a position-independent executable, which is position-independent code for address space layout randomization (ASLR)
- **Stack canaries:** Used for buffer overflow detection
- **RELRO:** Used for data section hardening
- **Source fortification:** Used for buffer overflow detection and format string protection
- **Stack clash protection:** Used for the detection of clashes between a stack pointer and another memory region
- **Control flow integrity (CFI) checks:** Used to prevent control flow hijacking
- **SafeStack:** Used for stack overflow protection

Compilers enable a few of these mitigations by default. For more detail on these exploit mitigation technologies, refer to appendix D: Compiler Mitigations.

In particular, the `checksec` tool reports that binaries produced by Temurin do not have stack canaries or source fortification enabled.

### Recommendations

Short term, enable security mitigations for Temurin builds by using the compiler and linker flags described in appendix D: Compiler Mitigations. These flags can be added using the `--with-extra-cflags` and `--with-extra-cxxflags` arguments during configuration.

While compilers often enable certain mitigations by default, if they are explicitly enabled, they will be used regardless of a compiler's defaults.

Long term, enable security mitigations for all binaries built by Temurin and add a scan for them into the test phase to ensure that certain options are always enabled. This will make it more difficult for an attacker to exploit any bugs found in the binaries.

**References**

- Airbus: Getting the maximum of your C compiler, for security
- Debian Hardening: Notes on Memory Corruption Mitigation Methods
- GCC Linux man page
- LD Linux man page
- OpenSSF's Compiler Options Hardening Guide for C and C++

## 15. Use of unpinned third-party workflows

| | |
|---|---|
| Severity: **Medium** | Difficulty: **High** |
| Type: Patching | Finding ID: TOB-TEMURIN-15 |
| Target: `temurin-build/.github/workflows/build-autotriage.yml`, `ci-jenkins-pipelines/.github/workflows/labeler.yml`, `infrastructure/.github/workflows/build_qemu.yml` | |

### Description

Workflows throughout the Temurin repositories directly use third-party workflows. Most of them are pinned to commit hashes, but there are some exceptions, such as in `ci-jenkins-pipelines/.github/workflows/labeler.yml`:

```
- uses: fuxingloh/multi-labeler@v2
  with:
    github-token: "${{secrets.GITHUB_TOKEN}}"
    config-path: .github/regex_labeler.yml
```

*Figure 15.1: Use of third-party workflow (`ci-jenkins-pipelines/.github/workflows/labeler.yml:19–22`)*

Git tags are malleable. This means that, while `fuxingloh/multi-labeler` is pinned to v2, the upstream may silently change the reference pointed to by v2. This can include malicious re-tags, in which case Temurin's various dependent workflows will silently update to the malicious workflow.

GitHub's security hardening guidelines for third-party actions encourage developers to pin third-party actions to a full-length commit hash. Generally excluded from this are "official" actions under the `actions` organization.

The following are the affected workflows:

- `temurin-build/.github/workflows/build-autotriage.yml`
- `ci-jenkins-pipelines/.github/workflows/labeler.yml`
- `infrastructure/.github/workflows/build_qemu.yml`

### Exploit Scenario

An attacker (or compromised maintainer) silently overwrites the v2 tag on `fuxingloh/multi-labeler` with a malicious version of the action, allowing the `secrets.GITHUB_TOKEN` value for the `ci-jenkins-pipeline` repository to be stolen.

## Recommendations

Short term, replace the current version tags with full-length commit hashes corresponding to the revision that each workflow is intended to use.

## 16. Third-party dependencies used without signature or checksum verification

| | |
|---|---|
| Severity: **Informational** | Difficulty: **High** |
| Type: Patching | Finding ID: TOB-TEMURIN-16 |
| Target: `temurin-build` | |

### Description

In many places in the `temurin-build` repository, third-party dependencies are installed via `https` download without a signature or checksum check. The following is a (not necessarily exhaustive) list of the dependencies that are installed in this way:

- In `tooling/linux_repro_build_compare.sh`:
    - `https://ftp.gnu.org/gnu/autoconf/autoconf-2.69.tar.gz`
    - `https://archive.apache.org/dist/ant/binaries/apache-ant-${ANT_VERSION}-bin.zip`
    - `https://sourceforge.net/projects/ant-contrib/files/ant-contrib/${ANT_CONTRIB_VERSION}/ant-contrib-${ANT_CONTRIB_VERSION}-bin.zip`
- In `tooling/release_download_test.sh`:
    - `https://github.com/CycloneDX/cyclonedx-cli/releases/download/v0.25.0/"${cyclonedx_tool}`
- In `build-farm/platform-specific-configurations/linux.sh`:
    - `https://github.com/alibaba/dragonwell8/releases/download/dragonwell-8.11.12_jdk8u332-ga/Alibaba_Dragonwell_8.11.12_x64_linux.tar.gz`
    - `https://github.com/alibaba/dragonwell8/releases/download/dragonwell-8.8.9_jdk8u302-ga/Alibaba_Dragonwell_8.8.9_aarch64_linux.tar.gz`
- In `.azure-devops/build/steps/windows/before.yml`:
    - `https://cygwin.com/setup-x86_64.exe`
- In `.github/workflows/build.yml`:

- `https://download.visualstudio.microsoft.com/download/pr/c5c75dfa-1b29-4419-80f8-bd39aed6bcd9/7ed8fa27575648163e07548ff5667b55b95663a2323e2b2a5f87b16284e481e6/vs_Community.exe`
- `https://download.visualstudio.microsoft.com/download/pr/6b655578-de8c-4862-ad77-65044ca714cf/f29399a618bd3a8d1dcc96d349453f686b6176590d904308402a6402543e310b/vs_Community.exe`

- In `docker/buildDocker.sh`:
  - `https://raw.githubusercontent.com/eclipse-openj9/openj9/master/buildenv/docker/mkdocker.sh`

### Recommendations
Short term, add a checksum or signature check to these downloads, wherever possible.

## 17. Code injection vulnerability in build-scripts pipeline jobs

| Severity: **High** | Difficulty: **High** |
|---|---|
| Type: Data Validation | Finding ID: TOB-TEMURIN-17 |
| Target: `ci-jenkins-pipelines/pipelines/build/common/build_base_file.groovy`, `ci-jenkins-pipelines/pipelines/build/common/openjdk_build_pipeline.groovy`, `ci-jenkins-pipelines/pipelines/build/openjdk_pipeline.groovy` | |

### Description

Jenkins pipeline jobs can execute arbitrary shell script code with the sh step. User input may reach sh calls through parameters or configurations originating from web-based form input. This allows for code injection and arbitrary code execution. The following sh calls receive input from external sources:

```
context.sh "rm -rf target/${config.TARGET_OS}/${config.ARCHITECTURE}/${config.VARIANT}/"
```

*Figure 17.1: TARGET_OS, ARCHITECTURE, and VARIANT input passed to sh (`ci-jenkins-pipelines/pipelines/build/common/build_base_file.groovy:898`)*

```
context.sh(script: "docker pull ${buildConfig.DOCKER_IMAGE} ${buildConfig.DOCKER_ARGS}")
...
context.sh(script: "docker pull ${buildConfig.DOCKER_IMAGE} ${buildConfig.DOCKER_ARGS}")
...
dockerImageDigest = context.sh(script: "docker inspect --format='{{.RepoDigests}}'
${buildConfig.DOCKER_IMAGE}", returnStdout:true)
```

*Figure 17.2: DOCKER_IMAGE and DOCKER_ARGS input passed to sh (`ci-jenkins-pipelines/pipelines/build/common/openjdk_build_pipeline.groovy:1915,1922,1928`)*

```
sh("curl -Os
https://raw.githubusercontent.com/adoptium/aqa-tests/${params.aqaReference}/testenv/
${propertyFile}")
```

*Figure 17.3: AQA_REF input passed to sh (`ci-jenkins-pipelines/pipelines/build/openjdk_pipeline.groovy:35`)*

If an attacker can influence any of these parameters, then they can execute arbitrary code on the Jenkins machine running the given job. The Eclipse Foundation has confirmed that these parameters can be specified in a Jenkins job web form; however, access to these forms is restricted.

### Exploit Scenario

An attacker sends a malicious shell payload through the `TARGET_OS`, `ARCHITECTURE`, `VARIANT`, `DOCKER_IMAGE`, `DOCKER_ARGS`, or `AQA_REF` Jenkins job parameters. The input then reaches the `sh` process, which allows the execution of arbitrary shell scripts. The attacker is able to execute arbitrary commands by using shell operators such as `;`, `&&`, or `||`, and to append additional commands.

### Recommendations

Short term, instead of specifying shell script commands to run in the `sh` step, use Groovy code or Jenkins plugins to accomplish the same action. For example, instead of `rm` or `curl`, use the `deleteDir` step or the File Operations plugin. Instead of using shell scripts for Docker operations, use the Docker Pipeline plugin where possible. If additional Docker command flags are necessary, use Boolean inputs that enable or disable specific flags instead of interpolating arbitrary string input.

Long term, implement static analysis rules to automatically detect when user input is passed to `sh` steps.

### References

- Jenkins, sh: Shell Script
- Docker Pipeline plugin, Advanced usage

## 18. Docker commands specify root user in containers

| | |
|---|---|
| Severity: **Informational** | Difficulty: **High** |
| Type: Access Controls | Finding ID: TOB-TEMURIN-18 |
| Target: `temurin-build/docker/buildDocker.sh` | |

### Description
Docker may specify a container user during the build process in a Dockerfile or at runtime on the command line. Running containers as root violates the principle of least privilege and should be avoided. The following Docker commands specify root as the container user:

```
docker run -it -u root -d --name="${dockerContainer}" "${dockerImage}"
docker exec -u root -i "${dockerContainer}" sh -c "git clone
https://github.com/ibmruntimes/openj9-openjdk-${jdk}"
docker exec -u root -i "${dockerContainer}" sh -c "cd openj9-openjdk-${jdk} && bash
./get_source.sh && bash ./configure --with-freemarker-jar=/root/freemarker.jar &&
make all"
```

*Figure 18.1: Commands specifying root container users (`temurin-build/docker/buildDocker.sh:141–143`)*

### Recommendations
Short term, have any necessary root actions performed at build-time in the Dockerfile, and have containers run as a lower privileged user at runtime.

Long term, once containers are no longer being run as root, enable the `--security-opt=no-new-privileges` flag when running Docker, in order to prevent privilege escalation using `setuid` or `setgid` binaries.

## 19. Incorrect Dependabot configuration filename

| | |
|---|---|
| Severity: **Undetermined** | Difficulty: **High** |
| Type: Patching | Finding ID: TOB-TEMURIN-19 |
| Target: `infrastructure/.github/dependabot` | |

### Description
The `infrastructure` repository has a Dependabot configuration file, used to configure the Dependabot bot, which detects out-of-date dependencies. However, this file is incorrectly named `dependabot` rather than `dependabot.yml`, preventing the bot from being run on this repository.

In order to test this, we created a private copy of the infrastructure repository and renamed the `dependabot` file to `dependabot.yml`. Dependabot detected many out-of-date Github Actions dependencies. We did not determine whether any of the out-of-date dependencies present in the `infrastructure` repository have security problems that could affect the Temurin infrastructure or build system.

### Recommendations
Short term, rename the `dependabot` file to `dependabot.yml`.

## A. Vulnerability Categories

The following tables describe the vulnerability categories, severity levels, and difficulty levels used in this document.

| Vulnerability Categories | |
|---|---|
| **Category** | **Description** |
| **Access Controls** | Insufficient authorization or assessment of rights |
| **Auditing and Logging** | Insufficient auditing of actions or logging of problems |
| **Authentication** | Improper identification of users |
| **Configuration** | Misconfigured servers, devices, or software components |
| **Cryptography** | A breach of system confidentiality or integrity |
| **Data Exposure** | Exposure of sensitive information |
| **Data Validation** | Improper reliance on the structure or values of data |
| **Denial of Service** | A system failure with an availability impact |
| **Error Reporting** | Insecure or insufficient reporting of error conditions |
| **Patching** | Use of an outdated software package or library |
| **Session Management** | Improper identification of authenticated users |
| **Testing** | Insufficient test methodology or test coverage |
| **Timing** | Race conditions or other order-of-operations flaws |
| **Undefined Behavior** | Undefined behavior triggered within the system |

| Severity Levels | |
|---|---|
| **Severity** | **Description** |
| **Informational** | The issue does not pose an immediate risk but is relevant to security best practices. |
| **Undetermined** | The extent of the risk was not determined during this engagement. |
| **Low** | The risk is small or is not one the client has indicated is important. |
| **Medium** | User information is at risk; exploitation could pose reputational, legal, or moderate financial risks. |
| **High** | The flaw could affect numerous users and have serious reputational, legal, or financial implications. |

| Difficulty Levels | |
|---|---|
| **Difficulty** | **Description** |
| **Undetermined** | The difficulty of exploitation was not determined during this engagement. |
| **Low** | The flaw is well known; public tools for its exploitation exist or can be scripted. |
| **Medium** | An attacker must write an exploit or will need in-depth knowledge of the system. |
| **High** | An attacker must have privileged access to the system, may need to know complex technical details, or must discover other weaknesses to exploit this issue. |

## B. Code Maturity Categories

The following tables describe the code maturity categories and rating criteria used in this document.

| Code Maturity Categories | |
|---|---|
| **Category** | **Description** |
| **Arithmetic** | The proper use of mathematical operations and semantics |
| **Auditing** | The use of event auditing and logging to support monitoring |
| **Authentication / Access Controls** | The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system |
| **Complexity Management** | The presence of clear structures designed to manage system complexity, including the separation of system logic into clearly defined functions |
| **Configuration** | The configuration of system components in accordance with best practices |
| **Cryptography and Key Management** | The safe use of cryptographic primitives and functions, along with the presence of robust mechanisms for key generation and distribution |
| **Data Handling** | The safe handling of user inputs and data processed by the system |
| **Documentation** | The presence of comprehensive and readable codebase documentation |
| **Maintenance** | The timely maintenance of system components to mitigate risk |
| **Memory Safety and Error Handling** | The presence of memory safety and robust error-handling mechanisms |
| **Testing and Verification** | The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage |

| Rating Criteria | |
|---|---|
| **Rating** | **Description** |
| **Strong** | No issues were found, and the system exceeds industry standards. |
| **Satisfactory** | Minor issues were found, but the system is compliant with best practices. |
| **Moderate** | Some issues that may affect system safety were found. |

| | |
|---|---|
| **Weak** | Many issues that affect system safety were found. |
| **Missing** | A required component is missing, significantly affecting system safety. |
| **Not Applicable** | The category is not applicable to this review. |
| **Not Considered** | The category was not considered in this review. |
| **Further Investigation Required** | Further investigation is required to reach a meaningful conclusion. |

## C. Insecure Download Semgrep Results

The following Semgrep results were produced by searching for insecure software downloads, including downloads over unencrypted channels such as HTTP and those in which SSL certificate validation is disabled.

```
infrastructure/ansible/playbooks/AdoptOpenJDK_AIX_Playbook/roles/yum/tasks/main.yml
   get-url-validate-certs-disabled
      Found file download with SSL verification disabled
       53┆ validate_certs: false

infrastructure/ansible/playbooks/AdoptOpenJDK_ITW_Playbook/roles/Common/tasks/CentOS.yml
   yum-validate-certs-disabled
      Found yum with SSL verification disabled
       14┆ validate_certs: no

infrastructure/ansible/playbooks/AdoptOpenJDK_Unix_Playbook/roles/Common/tasks/CentOS.yml
   yum-validate-certs-disabled
      Found yum with SSL verification disabled
       20┆ validate_certs: no

infrastructure/ansible/playbooks/AdoptOpenJDK_Unix_Playbook/roles/Common/tasks/Debian.yml
   apt-key-unencrypted-url
      Found apt key download with unencrypted URL (e.g. HTTP, FTP, etc.)
       63┆ url: http://repos.azulsystems.com/RPM-GPG-KEY-azulsystems

infrastructure/ansible/playbooks/AdoptOpenJDK_Unix_Playbook/roles/Common/tasks/SLES.yml
   get-url-validate-certs-disabled
      Found file download with SSL verification disabled
      222┆ validate_certs: no

infrastructure/ansible/playbooks/AdoptOpenJDK_Unix_Playbook/roles/Common/tasks/Ubuntu.yml
   apt-key-unencrypted-url
      Found apt key download with unencrypted URL (e.g. HTTP, FTP, etc.)
       27┆ url: http://repos.azulsystems.com/RPM-GPG-KEY-azulsystems

infrastructure/ansible/playbooks/AdoptOpenJDK_Unix_Playbook/roles/OpenSSL/tasks/main.yml
   get-url-validate-certs-disabled
      Found file download with SSL verification disabled
       62┆ validate_certs: no

infrastructure/ansible/playbooks/AdoptOpenJDK_Unix_Playbook/roles/freemarker/tasks/main.yml
   unarchive-validate-certs-disabled
      Found unarchive download with SSL verification disabled
       33┆ validate_certs: False
```

## D. Compiler Mitigations

The following table lists compiler security hardening flags available in modern compilers. Note that some of them can be enabled by default in a given compiler (like `--enable-default-pie` in GCC) and they may also influence the program's performance. We recommend reviewing those settings in order to harden production builds as much as possible.

| GCC or Clang Flag | What It Enables or Does |
|---|---|
| `Wl,-z,noexecstack` | This flag marks the program's data sections (including the stack and heap) as non-executable (NX).<br><br>This makes it more difficult for an attacker to execute shellcode. Attackers who wish to bypass NX must resort to return-oriented programming (ROP), an exploitation method that is more difficult and less reliable across different builds of a program. This mitigation is enabled by default. |
| `-Wl,-z,relro,-z,now` | This flag enables full RELRO (relocations read-only). Segments are read-only after relocation, and lazy bindings are disabled.<br><br>It is a mitigation technique used to harden the data sections of an ELF process. It has three modes of operation: disabled, partial, and full. When a program uses a function from a dynamically loaded library, the function address is stored in the `GOT.PLT` section.<br><br>When RELRO is disabled, each function address entry in the `GOT.PLT` table points to a dynamic resolver that resolves the entry to the actual address of the intended function when it is first called. In such a case, the memory location of the address is both readable and writable. As a result, an attacker who has control over the process control flow could change the entry of a given function in `GOT.PLT` to point to any other executable address. For example, the attacker could change the `puts` function's `GOT.PLT` entry to point to a system function. Then, if the program called `puts("bin/sh")`, `system("/bin/sh")` would be called instead. When RELRO is fully enabled, the dynamic resolver resolves all of the addresses upon a |

| | |
|---|---|
| | program's startup and changes the permissions of data sections (and therefore `GOT.PLT`) to read-only. |
| `-fstack-protector-all`<br><br>Or (less secure):<br><br>`-fstack-protector-strong --param ssp-buffer-size=4` | This flag adds stack canaries (stack cookies) for all functions. Note that this flag may affect the program's performance.<br><br>Stack canaries make it more difficult to exploit stack buffer overflow vulnerabilities. A stack canary is a global, randomly generated value that is copied to the stack between the stack variables and stack metadata in a function's prologue. When a function returns, the canary on the stack is checked against the global value. The program exits if there is a mismatch, making it more difficult for an attacker to overwrite the return address on the stack. In certain circumstances, attackers may be able to bypass this mitigation by disclosing the canary through a separate information disclosure vulnerability or by brute forcing the canary byte by byte.<br><br>To protect only functions that have buffers, use the alternative version of the flag indicated. |
| `-fPIE -pie` | This flag compiles the program as a position-independent executable, which address space layout randomization (ASLR), detailed below in the "System" rows, depends on. |
| Only in GCC >=12.x:<br><br>`-D_FORTIFY_SOURCE=3 -O2`<br><br>Or (less secure):<br><br>`-D_FORTIFY_SOURCE=2 -O2`<br><br>Or (even less secure):<br><br>`-D_FORTIFY_SOURCE=1 -O2` | This flag enables source fortification protections. These protections require an optimization flag (`-O1`, `-O2`, or `-O3`).<br><br>The protection is a libc-specific feature that enables a series of mitigations primarily aimed at preventing buffer overflows. It is supported by both glibc and Apple Libc, but not by musl or uclibc.<br><br>With a `_FORTIFY_SOURCE` level of 1, compile-time warnings are added for potentially unsafe calls to common libc functions (e.g., `memcpy` and `strcpy`). With a `_FORTIFY_SOURCE` level of 2, more stringent runtime checks are added to these functions and enable a number of lesser-known mitigations. For example, it will disallow the use of the `%n` format specifier in format strings that are not located in read-only memory pages. |

| | |
|---|---|
| | This will prevent overwriting data (and gaining code execution) with format string vulnerabilities.<br><br>The latter version is less secure, as it enables only compile-time measures; the former adds additional runtime checks, which may affect the program's performance.<br><br>The `_FORTIFY_SOURCE` level of 3 was added in GCC 12.x and further improves this feature's detection capabilities and coverage. |
| `-fstack-clash-protection` | This flag adds checks to functions that may allocate a large amount of memory on the stack to ensure that the new stack pointer and stack frame will not overlap with another memory region, such as the heap.<br><br>It mitigates a "stack clash vulnerability" in which a program's stack memory region grows so much that it overlaps with another memory region. This bug makes the program confuse the stack memory address with another memory address (e.g., that of the heap); as a result, the regions' data will overlap, which could lead to a denial of service or to control flow hijacking. The stack clash protection mitigation adds explicit memory probing to any function that allocates a large amount of stack memory; when explicit memory probing is used, the function's stack allocation will never make the stack pointer jump over the stack memory guard page, which is located before the stack. |
| `-fsanitize=cfi`<br>`-fvisibility=hidden`<br>`-flto`<br><br>(Clang/LLVM only) | This flag enables control flow integrity (CFI) checks that help prevent control flow hijacking. |
| `-fsanitize=safe-stack`<br><br>(Clang/LLVM only) | This flag enables SafeStack, which splits the stack frames of certain functions into a safe stack and an unsafe stack, making hijacking of the program's control flow more difficult (Clang/LLVM only). |
| `-Wall -Wextra -Wpedantic`<br>`-Wshadow -Wconversion` | These flags enable compile-time checks and warnings to detect potential problems in the code. |

|  |  |
|---|---|
| `-Wformat-security`<br>`-Wshorten-64-to-32` |  |

| System | What It Enables or Does |
|---|---|
| ASLR | This feature randomizes the memory location of each section of the program. This makes it more difficult for an attacker to write reliable exploits, primarily by impeding jumps to ROP gadgets. ASLR requires cooperation from both the system and the compiler.<br><br>To fully support ASLR, a program must be compiled as a position-independent executable. Most of the Linux distributions have ASLR enabled. This can be checked by reading the value stored in the `/proc/sys/kernel/randomize_va_space` file: `0` means that ASLR is disabled, 1 means it is partially enabled (only some bits of the addresses are randomized), and 2 means it is fully enabled. This file is writable, and an admin can disable or enable the mitigation. An information disclosure in the program may enable an attacker to bypass ASLR. |

## E. Code Quality Recommendations

This appendix contains findings that do not have immediate or obvious security implications or that were discovered but not fully investigated due to time constraints or scope limitations.

- **Java Virtual Machine (JVM) garbage collector manually invoked:** Calling `System.gc()` suggests to the JVM that the garbage collector should be run and memory should be reclaimed. This is only a suggestion; there is no guarantee that anything will happen. Relying on this behavior for correctness should be considered an anti-pattern. Note that this method is called only in test code. Nonetheless, it should not be relied on to enforce correct behavior. The API server calls this function in the following location:

```
override fun afterAll(p0: ExtensionContext?) {
    System.gc() // Don't ask, but also don't remove me, breaks deadlock that hangs
vm after all tests are completed
}
```

*Figure E.1: Call to `System.gc()`*
*(`api.adoptium.net/adoptium-frontend-parent/adoptium-api-v3-frontend/src/test/kotlin/net/adoptium/api/DbExtension.kt:13–15`)*

- **Dependencies hard-coded in Dockerfile:** Dependencies should instead be stored in a proper package management file, like `requirements.txt`, when building the Docker image. This allows a dependency scanner like Dependabot to automatically warn when dependencies have known vulnerabilities. Dependencies are hard-coded in the following location:

```
RUN pip install cryptography==2.9.2 PyYAML==5.3.1
```

*Figure E.2: Hard-coded Python pip dependencies*
*(`infrastructure/ansible/docker/Dockerfile.Ubuntu1604:15`)*

- **WinRM authentication missing TLS:** The current WinRM authentication configuration (CredSSP) is considered secure; however, best practice states that TLS should be used. Because TLS is disabled, WinRM server certificate validation is disabled in the following locations:

```
[windows:vars]
ansible_connection=winrm
ansible_port=5986
ansible_user=administrator
ansible_winrm_server_cert_validation=ignore
```

*Figure E.3: Server certificate validation disabled (`jenkins-helper/Jenkins_jobs/inventory-ini.template:27–32`)*

```
ansible_port: 5986
ansible_connection: winrm
ansible_winrm_server_cert_validation: ignore
```

*Figure E.4: Server certificate validation disabled (`infrastructure/ansible/playbooks/AdoptOpenJDK_Windows_Playbook/group_vars/all/adoptopenjdk_variables.yml:2–4`)*

- **Manual override of `in_container` check in Dockerfile:** Manually disabling this check when the code is in fact running in a container may have unintended consequences and cause unexpected behavior. This code is used to determine the configuration file location. Instead of modifying the code, use the correct configuration file location. This check is manually disabled in the following location:

```
RUN sed -i 's/\(def in_container():\)/\1\n    return False/g'
/usr/lib64/python*/*-packages/rhsm/config.py
```

*Figure E.5: Disabling of `in_container` check (`infrastructure/ansible/docker/Dockerfile.RHEL7:6`)*

- **Multiple third-party GitHub Actions used to make pull request comments:** There are two GitHub Actions used to make pull request comments: `JJ/pr-greeting-action` and `peter-evans/create-or-update-comment`. Furthermore, both of these actions use `pull_request_target`, which has known security weaknesses. To minimize the attack surface and reduce the risk of `pull_request_target` events, use a single GitHub Action to make pull request comments.

- **Broken link:** The `temurin-build/.azure-devops/build/steps/macOS/before.yml` file contains a broken link in a comment.

```
# install Xcode command line tools based on
#
https://github.com/AdoptOpenJDK/openjdk-infrastructure/blob/master/ansible/playbooks
/AdoptOpenJDK_Unix_Playbook/roles/Common/scripts/install-xcode.sh
- bash: |
```

*Figure E.6: `temurin-build/.azure-devops/build/steps/macOS/before.yml:20–22`*

- **Download from `api.adoptopenjdk.net`:** The `temurin-build/build-farm/platform-specific-configurations/linux.sh` file performs a download from `api.adoptopenjdk.net`, Adoptium's previous API URL before its name was changed.

```
# TOFIX: Switch this back once Semeru has an API to pull the nightly builds.
curl -L
"https://api.adoptopenjdk.net/v3/binary/latest/${JAVA_FEATURE_VERSION}/ga/linux/${NA
TIVE_API_ARCH}/jdk/openj9/normal/adoptopenjdk" | tar xpzf - --strip-components=1 -C
"$BUILDJDK"
```

*Figure E.7:* `temurin-build/build-farm/platform-specific-configurations/linux.sh:56–57`

- **Use of sudo without resetting cached credentials:** In the platform-specific configuration for macOS, the `sudo` command is used without first resetting the cached credentials by running `sudo -k`. This means that the configuration code may perform actions as the root user without receiving explicit permission from the user.

```
echo "[WARNING] You may be asked for your su user password, attempting to switch
Xcode version to ${XCODE_SWITCH_PATH}"
sudo xcode-select --switch "${XCODE_SWITCH_PATH}"
```

*Figure E.8:* `temurin-build/build-farm/platform-specific-configurations/mac.sh:85–86`

- **Commented keychain login code:** The `temurin-build/build-farm/platform-specific-configurations/mac.sh` file contains commented-out code whose purpose is to "Login to KeyChain." Temurin developers should uncomment or remove this code.

```
## Login to KeyChain
## shellcheck disable=SC2046
## shellcheck disable=SC2006
#security unlock-keychain -p `cat ~/.password` login.keychain-db
#rm -rf codesign-test && touch codesign-test
#codesign --sign "Developer ID Application: London Jamocha Community CIC"
codesign-test
#codesign -dvvv codesign-test
#export BUILD_ARGS="${BUILD_ARGS} --codesign-identity 'Developer ID Application:
London Jamocha Community CIC'"
```

*Figure E.9:* `temurin-build/build-farm/platform-specific-configurations/mac.sh:75–82`

- **Error not explicitly handled:** In the `temurin-build/sbin/prepareWorkspace.sh` file, an error during a GPG key download is ignored, instead of immediately returning an exit code (line 348). A failed download will still likely cause an error later in the script (line 352).

```
348        echo "ERROR: gpg recv-keys final attempt has failed. Will not try again."
349      fi
350    done
351    echo -e "5\ny\n" |  gpg --batch --command-fd 0 --expert --edit-key
```

```
"${ALSA_LIB_GPGKEYID}" trust;
    gpg --verify alsa-lib.tar.bz2.sig alsa-lib.tar.bz2 || exit 1
```

*Figure E.10:* `temurin-build/sbin/prepareWorkspace.sh:348–352`

- **Docker images run in writable filesystem:** Docker images are run with the read-only filesystem configuration option turned off. This makes it slightly easier for the process running in these images to be compromised. The images should be run with a read-only filesystem (using the `--read-only` flag), using volume mounts and temporary volume mounts in locations where write access is needed.

```
docker run -it -u root -d --name="${dockerContainer}" "${dockerImage}"
docker exec -u root -i "${dockerContainer}" sh -c "git clone
https://github.com/ibmruntimes/openj9-openjdk-${jdk}"
docker exec -u root -i "${dockerContainer}" sh -c "cd openj9-openjdk-${jdk} && bash
./get_source.sh && bash ./configure --with-freemarker-jar=/root/freemarker.jar &&
make all"
```

*Figure E.11:* `temurin-build/docker/buildDocker.sh:141–143`

# F. Fix Review Results

When undertaking a fix review, Trail of Bits reviews the fixes implemented for issues identified in the original report. This work involves a review of specific areas of the source code and system configuration, not comprehensive analysis of the system.

From June 10 to June 11, 2024, Trail of Bits reviewed the fixes and mitigations implemented by the Temurin team for the issues identified in this report. We reviewed each fix to determine its effectiveness in resolving the associated issue.

In summary, of the 19 issues described in this report, Temurin has resolved 12 issues, has partially resolved two issues, and has not resolved the remaining five issues. For additional information, please see the Detailed Fix Review Results below.

| ID | Title | Status |
|---|---|---|
| 1 | Command injection vulnerability in WinRM script | Unresolved |
| 2 | Docker Compose ports exposed on all interfaces | Resolved |
| 3 | Insecure installation of Xcode software | Resolved |
| 4 | Insecure software downloads in Ansible playbooks | Resolved |
| 5 | Signature verification disabled during software installation | Resolved |
| 6 | Missing integrity check in Dragonwell Dockerfile | Resolved |
| 7 | Hostname verification disabled on MongoDB client | Resolved |
| 8 | RHEL build image includes password | Resolved |
| 9 | Insecure downloads using wget command | Resolved |
| 10 | Hard-coded CA bundle keystore password | Unresolved |
| 11 | Hard-coded Vagrant VM password | Unresolved |

|  |  |  |
|---|---|---|
| 12 | Missing integrity or authenticity check in jcov script download | Partially Resolved |
| 13 | SSH client disables host key verification | Partially Resolved |
| 14 | Compiler mitigations are not enabled | Unresolved |
| 15 | Use of unpinned third-party workflows | Resolved |
| 16 | Third-party dependencies used without signature or checksum verification | Resolved |
| 17 | Code injection vulnerability in build-scripts pipeline jobs | Resolved |
| 18 | Docker commands specify root user in containers | Unresolved |
| 19 | Incorrect Dependabot configuration filename | Resolved |

## Detailed Fix Review Results

**TOB-TEMURIN-1: Command injection vulnerability in WinRM script**
Unresolved. The client provided the following context for this finding's fix status:

> *The job that runs this script has extremely controlled access, and anybody with permissions to exploit this already has direct machine access when required.*

**TOB-TEMURIN-2: Docker Compose ports exposed on all interfaces**
Resolved in PR #860. This PR changes the port descriptors from `27017:27017` and `8080:8080` to `127.0.0.1:27017:27017` and `127.0.0.1:8080:8080`, respectively, preventing these ports from being accessed from outside the localhost.

**TOB-TEMURIN-3: Insecure installation of Xcode software**
Resolved in PR #3282. This PR adds SHA-256 checksum checks on the relevant HTTP download results. It also fixes the OS X version check so that it takes the major version into account.

**TOB-TEMURIN-4: Insecure software downloads in Ansible playbooks**
Resolved in PR #3329. This PR changes various `http` links to `https` links and changes `validate_certs` values from `false` to `true`.

**TOB-TEMURIN-5: Signature verification disabled during software installation**
Resolved in PR #3355 and PR #3591. PR #3355 removes the statement highlighted in figure 5.2, which disables GPG verification, and changes the `gpgcheck` variables shown in figure 5.3 from `false` to `true`. PR #3591 removes the statement highlighted in figure 5.1, which disables GPG verification.

**TOB-TEMURIN-6: Missing integrity check in Dragonwell Dockerfile**
Resolved in PR #1000. This PR adds a SHA-256 checksum check after the download of the Dragonwell software.

**TOB-TEMURIN-7: Hostname verification disabled on MongoDB client**
Resolved in PR #993 and PR #1054. PR #993 changes the argument passed to the `invalidHostNameAllowed` function depending on the value of a `DISABLE_MONGO_HOST_CHECK` environment variable. If the variable is unset, it defaults to disabling hostname verification (i.e., `invalidHostNameAllowed(true)`). PR #1054 changes the default behavior of this value to enable hostname verification (i.e., `invalidHostNameAllowed(false)`).

The Temurin team told us that `DISABLE_MONGO_HOST_CHECK` is never set to `true` in production; however, we are not able to verify that this is the case.

**TOB-TEMURIN-8: RHEL build image includes password**
Resolved in PR #3320. This PR moves the `ROSIPW` variable into a Docker build secret.

**TOB-TEMURIN-9: Insecure downloads using wget command**
Resolved in PR #58 and PR #3363. PR #58 removes the `--no-check-certificate` flag applied to the `wget` command shown in figure 9.3. PR #3363 adds a SHA-256 checksum check to the downloads shown in figure 9.1 and changes the `Setup-QEMU-Images.md` documentation page, replacing the `wget` command shown in figure 9.2 with an instruction to download some FTP links "in a secure fashion."

**TOB-TEMURIN-10: Hard-coded CA bundle keystore password**
Unresolved. The client provided the following context for this finding's fix status:

> *The hardcoded password in this code is only used to allow the update/deployment of a new cacerts file, and is not used or available outside of these processes.*

**TOB-TEMURIN-11: Hard-coded Vagrant VM password**
Unresolved. The client provided the following context for this finding's fix status:

> *The job that runs this script has extremely controlled access, and anybody with permissions to exploit this already has direct machine access when required.*

**TOB-TEMURIN-12: Missing integrity or authenticity check in jcov script download**
Partially resolved in PR #877. This PR adds commands to download an MD5 checksum and compare the checksum against the MD5 hash of the ASM tools file. However, the checksum and the file are downloaded from the same source, so this adds only a minimal amount of security; an attacker who can replace the ASM tools file with a malicious file could also replace the MD5 hash file. We recommend comparing the ASM tools file with a fixed hash.

In addition, MD5 is not collision-resistant, so a more secure hash function such as SHA-256 should be used instead.

**TOB-TEMURIN-13: SSH client disables host key verification**
Partially resolved in PR #3526. This PR removes the files shown in figures 13.1 through 13.3. However, the issues shown in figures 13.4 through 13.6 are still present.

The client provided the following context for this finding's fix status:

> *These locations are considered benign because they are connecting to internal, short-lived, or local-only services.*

**TOB-TEMURIN-14: Compiler mitigations are not enabled**
Unresolved. The Temurin team has investigated the possibility of enabling compiler mitigations but has not yet enabled them.

**TOB-TEMURIN-15: Use of unpinned third-party workflows**
Resolved in PR #3321, PR #3597, and PR #872. These PRs pin the versions of Github Actions dependencies using full-length commit hashes.

**TOB-TEMURIN-16: Third-party dependencies used without signature or checksum verification**
Resolved in PR #3522 and PR #3781. PR #3522 adds a checksum verification to the download in `tooling/release_download_test.sh`, and PR #3781 adds checksum verifications to all the other code locations listed in the issue.

**TOB-TEMURIN-17: Code injection vulnerability in build-scripts pipeline jobs**
Resolved in PR #873. This PR adds a check that sanitizes the relevant variables before they are expanded.

**TOB-TEMURIN-18: Docker commands specify root user in containers**
Unresolved. The client provided the following context for this finding's fix status:

> *The scripts referenced in the issue are not used in the production of the Temurin JDK binary deliverables, and are provided as part of a development toolset.*

**TOB-TEMURIN-19: Incorrect Dependabot configuration filename**
Resolved in PR #3321. This PR renames the `dependabot` file to `dependabot.yml`.

## G. Fix Review Status Categories

The following table describes the statuses used to indicate whether an issue has been sufficiently addressed.

| Fix Status | |
|---|---|
| **Status** | **Description** |
| Undetermined | The status of the issue was not determined during this engagement. |
| Unresolved | The issue persists and has not been resolved. |
| Partially Resolved | The issue persists but has been partially resolved. |
| Resolved | The issue has been sufficiently resolved. |