
Note: To generate HTML and PDF versions of this guide, install pandoc and pdflatex (from TeX Live): `sh sudo dnf install -y pandoc texlive` Then use the CMake targets `docs-html` and `docs-pdf`.

Heimdall SBOM Generator - User Guide

Table of Contents

Introduction

Heimdall is a comprehensive Software Bill of Materials (SBOM) generation plugin that works seamlessly with both LLVM LLD and GNU Gold linkers. It automatically generates accurate SBOMs during the linking process, capturing all components that actually make it into your final binaries, including comprehensive DWARF debug information.

Key Features

- **Dual Linker Support**: Works with both LLVM LLD and GNU Gold linkers
- **Multiple SBOM Formats**: SPDX 2.3/3.0 and CycloneDX 1.4-1.6 with version selection
- **Enhanced DWARF Integration**: Extracts source files, functions, and compile units
- **Comprehensive Validation**: Built-in validation tools for standards compliance
- **Cross-Platform**: Native support for macOS and Linux
- **Performance Optimized**: Minimal overhead during linking

Getting Started

Prerequisites

- C++11 compatible compiler (GCC 4.8+ or Clang 3.3+ recommended)
- C++14, C++17, or C++23 compatible compiler for enhanced features
- CMake 3.16+
- OpenSSL development libraries
- LLVM 19 (for DWARF support)
- libelf (for ELF parsing)
- BFD (for Gold plugin)

Quick Installation

```
git clone https://github.com/heimdall-sbom/heimdall.git
cd heimdall
./build.sh
```

C++ Standard Support

Heimdall supports C++11, C++14, C++17, and C++23 standards:

C++11 (Minimum): - Basic functionality - Uses `.find() != std::string::npos` for string operations - Compatible with older compilers

C++14/17/23 (Enhanced features): - Uses modern features like `std::string::contains()` (C++23) - `starts_with`, `ends_with` (C++20+) - Future support for `std::format`, `std::print`, and more

Building with Different Standards

You can build Heimdall with any supported C++ standard:

```
# Default (C++23)
cmake -B build -DHEIMDALL_CPP_STANDARD=23
cmake --build build

# C++17
cmake -B build_cpp17 -DHEIMDALL_CPP_STANDARD=17
cmake --build build_cpp17

# C++14
cmake -B build_cpp14 -DHEIMDALL_CPP_STANDARD=14
cmake --build build_cpp14

# C++11
cmake -B build_cpp11 -DHEIMDALL_CPP_STANDARD=11
cmake --build build_cpp11
```

Installation

macOS Setup

```
# Install LLVM/LLD and other dependencies
brew install llvm cmake ninja openssl
export PATH="/opt/homebrew/opt/llvm/bin:$PATH"

# Build Heimdall (LLD plugin only)
./build.sh
```

Linux Setup

Ubuntu/Debian

```bash
# Install all dependencies including Gold linker
sudo apt-get update
sudo apt-get install -y \
    build-essential \
    cmake \
    ninja-build \
    binutils \
    binutils-dev \
    libssl-dev \
    libelf-dev \
    libgtest-dev \
    pkg-config

# Install LLVM 19 (recommended for full DWARF support)
wget -O - https://apt.llvm.org/llvm-snapshot.gpg.key | sudo apt-key add -
echo "deb http://apt.llvm.org/jammy/ llvm-toolchain-jammy-19 main" | sudo tee /etc/apt/sourc
sudo apt-get update
sudo apt-get install -y llvm-19-dev liblld-19-dev

# Build Heimdall with both LLD and Gold plugins
./build.sh
```

Fedora/RHEL/CentOS

```bash
sudo yum install -y \
    gcc-c++ \
    cmake \
    ninja-build \
    llvm-devel \
    lld-devel \
    binutils-gold \
    openssl-devel \
    pkgconfig \
    zlib-devel \
    llvm-googletest \
    binutils-devel

./build.sh
```

Build Options

```bash
# Debug build with sanitizers
./build.sh --debug --sanitizers
```

```
# Build only LLD plugin (macOS default)
./build.sh --no-gold

# Build only Gold plugin (Linux only)
./build.sh --no-lld

# Custom build directory
./build.sh --build-dir mybuild --install-dir myinstall
```

Basic Usage

Simple C Program

```
# Compile with debug information for DWARF extraction
gcc -g -c main.c -o main.o

# Using LLD with CycloneDX 1.6 (default)
ld.lld --plugin-opt=load:./heimdall-lld.dylib \
       --plugin-opt=sbom-output:myapp.cyclonedx.json \
       --plugin-opt=format:cyclonedx \
       main.o -o myapp

# Using LLD with SPDX 3.0 JSON (default)
ld.lld --plugin-opt=load:./heimdall-lld.dylib \
       --plugin-opt=sbom-output:myapp.spdx.json \
       --plugin-opt=format:spdx \
       main.o -o myapp

# Using Gold (Linux only)
ld.gold --plugin ./heimdall-gold.so \
        --plugin-opt sbom-output=myapp.cyclonedx.json \
        --plugin-opt format=cyclonedx \
        main.o -o myapp
```

Plugin Options

LLD Plugin Options

```
--plugin-opt=load:./heimdall-lld.dylib          # Load the plugin
--plugin-opt=sbom-output:output.json             # Output file path
--plugin-opt=format:cyclonedx                    # Format (cyclonedx, spdx)
--plugin-opt=cyclonedx-version:1.6               # CycloneDX version (1.4, 1.5, 1.6)
--plugin-opt=spdx-version:3.0                    # SPDX version (2.3, 3.0)
--plugin-opt=verbose                             # Enable verbose output
--plugin-opt=no-debug-info                       # Disable DWARF extraction
--plugin-opt=include-system-libs                 # Include system libraries
```

Gold Plugin Options (Linux only)

```
--plugin ./heimdall-gold.so                       # Load the plugin
--plugin-opt sbom-output=output.json              # Output file path
--plugin-opt format=cyclonedx                     # Format (cyclonedx, spdx)
--plugin-opt cyclonedx-version=1.6                # CycloneDX version (1.4, 1.5, 1.6)
--plugin-opt spdx-version=3.0                     # SPDX version (2.3, 3.0)
--plugin-opt verbose                              # Enable verbose output
--plugin-opt no-debug-info                        # Disable DWARF extraction
--plugin-opt include-system-libs                  # Include system libraries
```

Advanced Features

Version Selection

Heimdall supports multiple versions of both SPDX and CycloneDX standards:

```
# CycloneDX versions
--plugin-opt=cyclonedx-version:1.4                # CycloneDX 1.4
--plugin-opt=cyclonedx-version:1.5                # CycloneDX 1.5
--plugin-opt=cyclonedx-version:1.6                # CycloneDX 1.6 (default)

# SPDX versions
--plugin-opt=spdx-version:2.3                     # SPDX 2.3 tag-value
--plugin-opt=spdx-version:3.0                     # SPDX 3.0 JSON (default)
```

Verbose Output

Enable detailed processing information:

```
ld.lld --plugin-opt=load:./heimdall-lld.dylib \
       --plugin-opt=sbom-output:debug-sbom.json \
       --plugin-opt=verbose \
       main.o -o debug-app
```

System Library Inclusion

Include system libraries in the SBOM:

```
ld.lld --plugin-opt=load:./heimdall-lld.dylib \
       --plugin-opt=sbom-output:full-sbom.json \
       --plugin-opt=include-system-libs \
       main.o -o full-app
```

SBOM Formats and Versions

CycloneDX 1.4-1.6

CycloneDX is a lightweight SBOM standard designed for application security contexts and supply chain component analysis.

Features by Version **CycloneDX 1.4:** - Basic component metadata - Dependencies and relationships - License information - Hash information

CycloneDX 1.5: - Enhanced metadata - External references - PURL identifiers - Improved relationships

CycloneDX 1.6 (Default): - Lifecycle information - Evidence collection - Enhanced properties - Extended metadata

Example CycloneDX Output

```
{
  "bomFormat": "CycloneDX",
  "specVersion": "1.6",
  "version": 1,
  "metadata": {
    "timestamp": "2024-01-15T10:30:00Z",
    "tools": [
      {
        "vendor": "Heimdall",
        "name": "SBOM Generator",
        "version": "1.0.0"
      }
    ]
  },
  "components": [
    {
      "type": "application",
      "name": "myapp",
      "version": "1.0.0",
      "purl": "pkg:generic/myapp@1.0.0",
      "properties": [
        {
          "name": "heimdall:source-files",
          "value": "main.c,utils.c"
        },
        {
          "name": "heimdall:functions",
          "value": "main,calculate,process_data"
        },
        {
          "name": "heimdall:contains-debug-info",
          "value": "true"
        }
      ]
    }
  ]
}
```

SPDX 2.3 and 3.0

SPDX is a comprehensive standard for communicating software bill of materials information.

Features by Version **SPDX 2.3:** - Tag-value format - Package and file information - License information - Relationships

SPDX 3.0 (Default): - JSON format - Enhanced relationships - Extended metadata - Improved structure

Example SPDX Output **SPDX 2.3 (Tag-Value):**

```
SPDXVersion: SPDX-2.3
DataLicense: CC0-1.0
SPDXID: SPDXRef-DOCUMENT
DocumentName: myapp-sbom
DocumentNamespace: https://example.com/myapp-sbom
Creator: Tool: Heimdall-SBOM-Generator-1.0.0
Created: 2024-01-15T10:30:00Z

PackageName: myapp
SPDXID: SPDXRef-myapp
PackageVersion: 1.0.0
PackageDownloadLocation: NOASSERTION
FilesAnalyzed: true
PackageLicenseConcluded: MIT
PackageLicenseDeclared: MIT

FileName: main.c
SPDXID: SPDXRef-main.c
FileChecksum: SHA256: abc123...
LicenseConcluded: MIT

Relationship: SPDXRef-myapp GENERATED_FROM SPDXRef-main.c
```

SPDX 3.0 (JSON):

```json
{
  "spdxVersion": "SPDX-3.0",
  "dataLicense": "CC0-1.0",
  "SPDXID": "SPDXRef-DOCUMENT",
  "name": "myapp-sbom",
  "documentNamespace": "https://example.com/myapp-sbom",
  "creationInfo": {
    "creators": ["Tool: Heimdall-SBOM-Generator-1.0.0"],
    "created": "2024-01-15T10:30:00Z"
  },
  "packages": [
    {
      "SPDXID": "SPDXRef-myapp",
      "name": "myapp",
      "versionInfo": "1.0.0",
      "downloadLocation": "NOASSERTION",
      "licenseConcluded": "MIT",
      "licenseDeclared": "MIT"
    }
  ],
  "files": [
    {
      "SPDXID": "SPDXRef-main.c",
      "fileName": "main.c",
      "checksums": [
        {
          "algorithm": "SHA256",
          "checksumValue": "abc123..."
        }
      ],
      "licenseConcluded": "MIT"
    }
  ],
  "relationships": [
    {
      "spdxElementId": "SPDXRef-myapp",
      "relatedSpdxElement": "SPDXRef-main.c",
      "relationshipType": "GENERATED_FROM"
    }
  ]
}
```

DWARF Debug Information

Heimdall provides comprehensive DWARF debug information integration that enhances your SBOMs with detailed source code traceability.

Source File Tracking

Heimdall automatically extracts source file paths from DWARF debug info and includes them as separate SBOM components:

```bash
# Compile with debug information
gcc -g -c main.c utils.c -o main.o
```

```
# Generate SBOM with source files
ld.lld --plugin-opt=load:./heimdall-lld.dylib \
       --plugin-opt=sbom-output:with-sources.json \
       --plugin-opt=format:cyclonedx \
       main.o -o myapp
```

Function-Level Analysis

Heimdall extracts function names and includes them in SBOM properties:

```
{
  "properties": [
    {
      "name": "heimdall:functions",
      "value": "main,calculate,process_data,validate_input"
    }
  ]
}
```

Compile Unit Information

Compile unit information is extracted for build system integration:

```
{
  "properties": [
    {
      "name": "heimdall:compile-units",
      "value": "main.c,utils.c"
    }
  ]
}
```

Fallback Extraction

When DWARF parsing fails, Heimdall uses ELF symbol table fallback:

```
# Even without DWARF, functions are extracted from symbols
gcc -c main.c -o main.o  # No debug info

ld.lld --plugin-opt=load:./heimdall-lld.dylib \
       --plugin-opt=sbom-output:fallback.json \
       main.o -o myapp
```

DWARF Properties

Heimdall adds custom properties to CycloneDX SBOMs:

- `heimdall:source-files`: Comma-separated list of source files
- `heimdall:functions`: Comma-separated list of function names

- `heimdall:compile-units`: Comma-separated list of compile units
- `heimdall:contains-debug-info`: Boolean indicating DWARF presence

SBOM Validation and Tools

Heimdall includes comprehensive validation tools to ensure your SBOMs are standards compliant.

Command-Line Validation Tool

```
# Validate a single SBOM
./build/bin/heimdall-validate --validate myapp.cyclonedx.json
./build/bin/heimdall-validate --validate myapp.spdx.json

# Compare two SBOMs
./build/bin/heimdall-validate --compare sbom1.json sbom2.json

# Generate diff report
./build/bin/heimdall-validate --diff sbom1.json sbom2.json --output diff-report.txt

# Merge multiple SBOMs
./build/bin/heimdall-validate --merge sbom1.json sbom2.json sbom3.json --output merged.json
```

Validation Scripts

```
# Basic validation (JSON syntax, SPDX structure)
./scripts/validate_sboms.sh build

# Advanced validation (schema validation, detailed reporting)
python3 scripts/validate_sboms_online.py build
```

Validation Features

- **JSON syntax validation** for CycloneDX and SPDX 3.0 files
- **SPDX structure validation** (supports both 2.3 tag-value and 3.0 JSON formats)
- **Required field checking** for both standards
- **Detailed validation logs** and summary reports
- **SBOM comparison** and diff generation
- **SBOM merging** capabilities

CI/CD Integration

```
# GitHub Actions example
- name: Generate SBOMs
  run: |
    make build-test-sbom
```

```yaml
- name: Validate SBOMs
  run: |
    ./scripts/validate_sboms.sh build

- name: Upload SBOMs
  uses: actions/upload-artifact@v3
  with:
    name: sboms
    path: build/sboms/
```

Build System Integration

CMake Integration

Basic Integration

```cmake
# Find Heimdall
find_library(HEIMDALL_LLD heimdall-lld REQUIRED)

# Add SBOM generation to target
target_link_options(myapp PRIVATE
    "LINKER:--plugin-opt=load:${HEIMDALL_LLD}"
    "LINKER:--plugin-opt=sbom-output:${CMAKE_BINARY_DIR}/myapp-sbom.json"
    "LINKER:--plugin-opt=format:cyclonedx"
)

# Enable debug information for DWARF extraction
target_compile_options(myapp PRIVATE -g)
```

Advanced Integration with Version Selection

```cmake
# Function to add SBOM generation
function(add_sbom_generation target)
    find_library(HEIMDALL_LLD heimdall-lld REQUIRED)

    # CycloneDX 1.6 (default)
    target_link_options(${target} PRIVATE
        "LINKER:--plugin-opt=load:${HEIMDALL_LLD}"
        "LINKER:--plugin-opt=sbom-output:${CMAKE_BINARY_DIR}/${target}-cyclonedx.json"
        "LINKER:--plugin-opt=format:cyclonedx"
        "LINKER:--plugin-opt=cyclonedx-version:1.6"
    )

    # SPDX 3.0 JSON
    target_link_options(${target} PRIVATE
        "LINKER:--plugin-opt=load:${HEIMDALL_LLD}"
        "LINKER:--plugin-opt=sbom-output:${CMAKE_BINARY_DIR}/${target}-spdx.json"
        "LINKER:--plugin-opt=format:spdx"
        "LINKER:--plugin-opt=spdx-version:3.0"
    )

    # Enable debug information
    target_compile_options(${target} PRIVATE -g)
endfunction()

# Use the function
add_executable(myapp main.c)
add_sbom_generation(myapp)
```

Conditional Integration

```
# Only add SBOM generation if Heimdall is available
find_library(HEIMDALL_LLD heimdall-lld)
if(HEIMDALL_LLD)
    target_link_options(myapp PRIVATE
        "LINKER:--plugin-opt=load:${HEIMDALL_LLD}"
        "LINKER:--plugin-opt=sbom-output:${CMAKE_BINARY_DIR}/myapp-sbom.json"
        "LINKER:--plugin-opt=format:cyclonedx"
    )
    target_compile_options(myapp PRIVATE -g)
    message(STATUS "SBOM generation enabled for myapp")
else()
    message(WARNING "Heimdall not found, SBOM generation disabled")
endif()
```

Makefile Integration

Basic Makefile

```
CC = gcc
LD = ld.lld
HEIMDALL_PLUGIN = ./heimdall-lld.dylib

CFLAGS = -g -Wall
LDFLAGS = --plugin-opt=load:$(HEIMDALL_PLUGIN) \
          --plugin-opt=sbom-output:$(TARGET)-sbom.json \
          --plugin-opt=format:cyclonedx

TARGET = myapp
SOURCES = main.c utils.c
OBJECTS = $(SOURCES:.c=.o)

$(TARGET): $(OBJECTS)
```

```
    $(LD) $(LDFLAGS) $^ -o $@

%.o: %.c
    $(CC) $(CFLAGS) -c $< -o $@

clean:
    rm -f $(OBJECTS) $(TARGET) $(TARGET)-sbom.json
```

Advanced Makefile with Multiple Formats

```
CC = gcc
LD = ld.lld
HEIMDALL_PLUGIN = ./heimdall-lld.dylib

CFLAGS = -g -Wall
TARGET = myapp
SOURCES = main.c utils.c
OBJECTS = $(SOURCES:.c=.o)

# SBOM generation targets
.PHONY: sbom-cyclonedx sbom-spdx sbom-all

$(TARGET): $(OBJECTS)
    $(LD) --plugin-opt=load:$(HEIMDALL_PLUGIN) \
          --plugin-opt=sbom-output:$(TARGET)-cyclonedx.json \
          --plugin-opt=format:cyclonedx \
          --plugin-opt=cyclonedx-version:1.6 \
          $^ -o $@

sbom-cyclonedx: $(TARGET)
    @echo "Generated CycloneDX 1.6 SBOM: $(TARGET)-cyclonedx.json"

sbom-spdx: $(OBJECTS)
    $(LD) --plugin-opt=load:$(HEIMDALL_PLUGIN) \
          --plugin-opt=sbom-output:$(TARGET)-spdx.json \
          --plugin-opt=format:spdx \
          --plugin-opt=spdx-version:3.0 \
          $^ -o $(TARGET)-spdx
    @echo "Generated SPDX 3.0 SBOM: $(TARGET)-spdx.json"

sbom-all: sbom-cyclonedx sbom-spdx
    @echo "Generated all SBOM formats"

%.o: %.c
    $(CC) $(CFLAGS) -c $< -o $@
```

```
clean:
    rm -f $(OBJECTS) $(TARGET) $(TARGET)-spdx *.json
```

Meson Integration

```
# meson.build
project('myapp', 'c')

# Find Heimdall
heimdall_dep = dependency('heimdall-lld', required: false)

# Create executable
myapp = executable('myapp', 'main.c', 'utils.c',
    c_args: ['-g'],  # Enable debug info for DWARF
    link_args: heimdall_dep.found() ? [
        '--plugin-opt=load:' + heimdall_dep.get_variable('plugin_path'),
        '--plugin-opt=sbom-output=@OUTPUT@-sbom.json',
        '--plugin-opt=format=cyclonedx'
    ] : []
)

# Add custom target for SBOM validation
if heimdall_dep.found()
    run_target('validate-sbom',
        command: ['heimdall-validate', '--validate', myapp.full_path() + '-sbom.json']
    )
endif()
```

Command-Line Integration

Shell Scripts

Simple Build Script

```
#!/bin/bash
# build-with-sbom.sh

set -e

TARGET=${1:-myapp}
FORMAT=${2:-cyclonedx}
VERSION=${3:-1.6}

echo "Building $TARGET with $FORMAT $VERSION SBOM..."

# Compile with debug info
gcc -g -c *.c
```

```bash
# Link with SBOM generation
if [ "$FORMAT" = "cyclonedx" ]; then
    ld.lld --plugin-opt=load:./heimdall-lld.dylib \
           --plugin-opt=sbom-output:$TARGET-$FORMAT.json \
           --plugin-opt=format:$FORMAT \
           --plugin-opt=cyclonedx-version:$VERSION \
           *.o -o $TARGET
elif [ "$FORMAT" = "spdx" ]; then
    ld.lld --plugin-opt=load:./heimdall-lld.dylib \
           --plugin-opt=sbom-output:$TARGET-$FORMAT.json \
           --plugin-opt=format:$FORMAT \
           --plugin-opt=spdx-version:$VERSION \
           *.o -o $TARGET
fi

echo "Generated $TARGET-$FORMAT.json"
```

Advanced Build Script with Validation

```bash
#!/bin/bash
# build-and-validate.sh

set -e

TARGET=${1:-myapp}
VALIDATE=${2:-true}

echo "Building $TARGET..."

# Build with both formats
./build-with-sbom.sh $TARGET cyclonedx 1.6
./build-with-sbom.sh $TARGET spdx 3.0

if [ "$VALIDATE" = "true" ]; then
    echo "Validating SBOMs..."

    # Validate CycloneDX
    if ./build/bin/heimdall-validate --validate $TARGET-cyclonedx.json; then
        echo "  CycloneDX SBOM is valid"
    else
        echo "  CycloneDX SBOM validation failed"
        exit 1
    fi

    # Validate SPDX
    if ./build/bin/heimdall-validate --validate $TARGET-spdx.json; then
        echo "  SPDX SBOM is valid"
    else
        echo "  SPDX SBOM validation failed"
        exit 1
    fi

    # Compare formats
    echo "Comparing SBOM formats..."
    ./build/bin/heimdall-validate --compare $TARGET-cyclonedx.json $TARGET-spdx.json
fi

echo "Build complete!"
```

Python Integration

Build Script with Python

```
#!/usr/bin/env python3
# build_sbom.py

import subprocess
import sys
import json
from pathlib import Path

def run_command(cmd, check=True):
    """Run a command and return the result."""
    result = subprocess.run(cmd, shell=True, capture_output=True, text=True)
    if check and result.returncode != 0:
        print(f"Command failed: {cmd}")
        print(f"Error: {result.stderr}")
        sys.exit(1)
    return result

def build_with_sbom(target, format_type="cyclonedx", version="1.6"):
    """Build a target with SBOM generation."""
    print(f"Building {target} with {format_type} {version}...")

    # Compile with debug info
    run_command("gcc -g -c *.c")

    # Build SBOM command
    if format_type == "cyclonedx":
        cmd = f"""ld.lld --plugin-opt=load:./heimdall-lld.dylib \\
                --plugin-opt=sbom-output:{target}-{format_type}.json \\
                 --plugin-opt=format:{format_type} \\
                 --plugin-opt=cyclonedx-version:{version} \\
                 *.o -o {target}"""
    elif format_type == "spdx":
        cmd = f"""ld.lld --plugin-opt=load:./heimdall-lld.dylib \\
                 --plugin-opt=sbom-output:{target}-{format_type}.json \\
                 --plugin-opt=format:{format_type} \\
                 --plugin-opt=spdx-version:{version} \\
                 *.o -o {target}"""

    run_command(cmd)
    print(f"Generated {target}-{format_type}.json")

def validate_sbom(sbom_file):
    """Validate an SBOM file."""
    print(f"Validating {sbom_file}...")
    result = run_command(f"./build/bin/heimdall-validate --validate {sbom_file}", check=Fals
    if result.returncode == 0:
        print(f"  {sbom_file} is valid")
        return True
    else:
        print(f"  {sbom_file} validation failed")
        return False

def main():
    target = sys.argv[1] if len(sys.argv) > 1 else "myapp"

    # Build with both formats
    build_with_sbom(target, "cyclonedx", "1.6")
    build_with_sbom(target, "spdx", "3.0")

    # Validate SBOMs
    cyclonedx_file = f"{target}-cyclonedx.json"
    spdx_file = f"{target}-spdx.json"

    cyclonedx_valid = validate_sbom(cyclonedx_file)
    spdx_valid = validate_sbom(spdx_file)

    if not (cyclonedx_valid and spdx_valid):
        sys.exit(1)

    print("Build and validation complete!")

if __name__ == "__main__":
    main()
```

Troubleshooting

Common Issues

Plugin Not Found

```
Error: Could not load plugin
```

Solution: Ensure the plugin path is correct and the plugin is built for your platform.

Missing Dependencies

```
Error: LLVM libraries not found
```

Solution: Install LLVM development packages.

Gold Linker Not Found (Linux)

```
Warning: Gold linker not found
```

Solution: Install `binutils-gold` package for your distribution.

Gold Not Available on macOS

```
Warning: Gold linker not found
```

Solution: This is expected on macOS. Use LLD linker instead.

Permission Denied

```
Error: Cannot write to output file
```

Solution: Check write permissions for the output directory.

OpenSSL Deprecation Warnings

```
Warning: 'SHA256_Init' is deprecated
```

Solution: These warnings are harmless and the build will succeed.

DWARF Extraction Not Working

```
Warning: No DWARF debug information found
```

Solution: Ensure you're compiling with debug information (`-g` flag) and using LLVM 19+ for best DWARF support.

Platform-Specific Issues

macOS

- **Gold plugin not built**: This is expected behavior. Gold is not available on macOS.
- **LLD path issues**: Ensure LLVM is properly installed via Homebrew and PATH is set correctly.
- **DWARF support**: Full DWARF support available with Homebrew LLVM.

Linux

- **Gold not found**: Install `binutils-gold` package for your distribution.
- **Plugin loading errors**: Ensure Gold was built with plugin support enabled.
- **DWARF support**: Full DWARF support available with LLVM 19+.

Debug Mode

Enable verbose output to see detailed processing information:

```
# LLD
ld.lld --plugin-opt=load:./heimdall-lld.dylib \
       --plugin-opt=sbom-output:debug-sbom.json \
       --plugin-opt=verbose \
       main.o -o debug-app

# Gold (Linux only)
ld.gold --plugin ./heimdall-gold.so \
        --plugin-opt sbom-output=debug-sbom.json \
        --plugin-opt verbose \
        main.o -o debug-app
```

Logging Levels

Heimdall provides different log levels:

- **INFO**: General processing information
- **WARNING**: Non-critical issues
- **ERROR**: Critical errors
- **DEBUG**: Detailed debugging information (requires debug build)

Best Practices

SBOM Generation

1. **Always compile with debug information** for comprehensive DWARF extraction:

```
gcc -g -c source.c -o source.o
```

2. **Use appropriate SBOM format versions** for your toolchain:

```
# For modern tools
--plugin-opt=cyclonedx-version:1.6
--plugin-opt=spdx-version:3.0

# For legacy compatibility
--plugin-opt=cyclonedx-version:1.4
--plugin-opt=spdx-version:2.3
```

3. **Include system libraries** when needed for complete dependency tracking:

```
--plugin-opt=include-system-libs
```

4. **Validate generated SBOMs** to ensure standards compliance:

```
./build/bin/heimdall-validate --validate myapp.cyclonedx.json
```

Build System Integration

1. **Use conditional integration** to handle missing Heimdall gracefully:

```
find_library(HEIMDALL_LLD heimdall-lld)
if(HEIMDALL_LLD)
    # Add SBOM generation
endif()
```

2. **Enable debug information** in your build system:

```
target_compile_options(myapp PRIVATE -g)
```

3. **Generate multiple formats** for maximum compatibility:

```
# Generate both CycloneDX and SPDX
add_sbom_generation(myapp)
```

CI/CD Integration

1. **Validate SBOMs in CI/CD** to catch issues early:

```
- name: Validate SBOMs
  run: ./scripts/validate_sboms.sh build
```

2. **Upload SBOMs as artifacts** for later analysis:

```
- name: Upload SBOMs
  uses: actions/upload-artifact@v3
  with:
    name: sboms
    path: build/sboms/
```

3. **Compare SBOMs across builds** to track changes:

   ```
   ./build/bin/heimdall-validate --diff previous.json current.json
   ```

Performance Optimization

1. **Use appropriate linker** for your platform:
   - LLD: Better on macOS, good on Linux
   - Gold: Better on Linux, not available on macOS
2. **Disable verbose output** in production builds:

   ```
   # Remove --plugin-opt=verbose for production
   ```

3. **Cache build artifacts** to avoid regenerating SBOMs unnecessarily.

Security Considerations

1. **Review generated SBOMs** for sensitive information before distribution.
2. **Validate SBOMs** against known vulnerabilities:

   ```
   # Use external tools for vulnerability scanning
   grype sbom:./myapp.cyclonedx.json
   ```

3. **Sign SBOMs** when distributing to ensure integrity.

Documentation

1. **Document your SBOM generation process** in your project README.
2. **Include SBOM examples** in your documentation.
3. **Provide validation scripts** for your team to use.
4. **Update CI/CD documentation** to include SBOM generation steps.

For more information, see the main README and API documentation.

Requirements

- C++11 compatible compiler (GCC 4.8+ or Clang 3.3+ recommended)
- C++14, C++17, or C++23 compatible compiler for enhanced features
- CMake 3.16+
- OpenSSL
- LLVM 19 (for DWARF support)
- libelf (for ELF parsing)
- BFD (for Gold plugin)

C++ Standard Support

Heimdall supports multiple C++ standards with different feature sets:

C++11 (Minimum): - Basic functionality - Uses `.find() != std::string::npos` for string operations - Compatible with older compilers

C++14/17/23 (Enhanced features): - Uses modern features like `std::string::contains()` (C++23) - `starts_with`, `ends_with` (C++20+) - Future support for `std::format`, `std::print`, and more

If you encounter build errors, ensure your compiler supports the selected C++ standard and that CMake is passing the correct `-std=c++XX` flag to your compiler.