C/C++のシステムに対するSBOM生成手法の検討

音田 渉　神田 哲也　眞鍋 雄貴　井上 克郎　仇 実　肥後 芳樹

現在のソフトウェア開発では多くの外部ライブラリを活用するが，セキュリティや著作権上のリスクが伴う．この問題に対処するためにSBOMの活用が奨励されているが，C/C++で開発されたシステムに対するSBOM生成技術が確立できていないため，ビルド処理と成果物であるバイナリから得られる情報に着目して生成手法を検討した．

1　ソフトウェア部品表（SBOM）

開発効率向上や高機能化などのために外部ライブラリを用いたソフトウェア開発が広く行われている一方，セキュリティや著作権上のリスクが伴う [2]．これらのリスクは早期に発見し対応することが重要である．しかし，現在のソフトウェア開発で利用する外部ライブラリの数は，その外部ライブラリが依存する別のライブラリのような推移的な依存関係も考慮すると膨大なものとなり [3]，適切な管理は難しい．

この問題に対処するためにソフトウェア部品表（Software Bill of Materials, SBOM）の活用が奨励されている．SBOMは，ソフトウェアの構築に用いられるライブラリなどの部品の正式かつ機械可読な表であり，各部品のライセンス・バージョン・ベンダなどの詳細や，部品間のサプライチェーン関係の情報を含む [5]．米国をはじめ各国の政府機関がSBOM利用を推進する [1] など，近年急速に注目が集まっている．しかし，普及にあたってはSBOM利活用に必要なツールの不足が障壁となっている [4]．特に，npmやpipといったパッケージマネージャの情報を用いてSBOMを生成するツールは流通しているものの，パッケージマネージャが普及していないC/C++で開発されたシステムについては，現状SBOMを容易に生成する方法が存在しない．

2　C/C++における依存関係抽出

我々はビルド処理から得られる情報とその結果生成されるバイナリに含まれるメタデータに着目する．手法の概要を図1に示す．まず，ビルド処理においてC/C++ではコンパイルの後にリンク処理が存在し，ここではプログラムが参照するシンボル情報とライブラリ内のシンボル情報の照合を行う．そして，静的ライブラリについてはビルド成果物に埋め込み，動的ラ

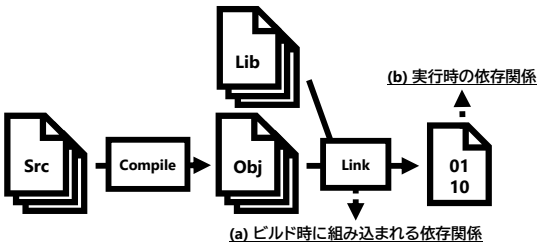

図1　手法概要

Investigating Techniques to Produce an SBOM for C/C++ Systems
Wataru Otoda, Yoshiki Higo, 大阪大学, Osaka University.
Tetsuya Kanda, ノートルダム清心女子大学, Notre Dame Seishin University.
Yuki Manabe, 福知山公立大学, The University of Fukuchiyama.
Katsuro Inoue, 南山大学, Nanzan University.
Shi Qiu, 株式会社東芝, Toshiba Corporation.

```
/usr/lib/x86_64-linux-gnu/libssl.so
/usr/lib/x86_64-linux-gnu/libcrypto.so
/usr/lib/gcc/x86_64-linux-gnu/13/libgcc.a
```

図 2　ビルド時に組み込まれる依存関係（抜粋）

イブラリについては必要なライブラリファイルとシンボルの情報をメタデータとして記録する．以上を踏まえると，(a) リンク処理時のリンカの動作を観測すればビルド時に組み込まれる依存関係を，(b) ビルド成果物バイナリのメタデータを解析すれば実行時の依存関係を，それぞれ取得できると考えられる．なお，以下の実験は Ubuntu 24.04 (AMD64) で実施した．

2.1　ビルド時に組み込まれる依存関係の抽出

C 言語で開発されている curl 8.10.1[†1] に対し，OpenSSL ライブラリを使用する`--with-openssl` をビルドオプションに加えた状態でリンカに動作ログを出力させたところ，図 2 のとおり OpenSSL と libgcc が依存に含まれることを確認できた．`.so` ファイルは shared object を表し実行時に読み込まれるものである．一方，`.a` ファイルは archive を表しリンク時に直接埋め込まれるものである．

しかし，SBOM の用途として脆弱性管理を考えるとファイルパスだけでは不足であり，少なくとも脆弱性の有無を決める要因となるライブラリ名とバージョンは含めたい．そのためには，シンボリックリンクが指す先を解析する，ファイルパスから推定する，pkg-config や OS 側のパッケージマネージャの情報を用いるなどの工夫が必要となる．さらに，実際の出力は内部のオブジェクトファイルなども混ざった大量のノイズを含むため，外部ライブラリに該当するものだけを選別する工夫も必要となる．

2.2　実行時の依存関係の抽出

前節で得られた実行可能バイナリに記録された依存情報を抽出したところ，図 3 のとおり同様の依存を確認できた．libgcc はリンク時に直接埋め込まれているためここでは抽出されない．これ以外にも，shared object が依存する shared object のような推移的な依存関係も取得できた．ただし，有用な SBOM を得るには前節と同様に工夫を要する．

```
/lib/x86_64-linux-gnu/libssl.so.3
/lib/x86_64-linux-gnu/libcrypto.so.3
```

図 3　実行時の依存関係（抜粋）

3　おわりに

本研究では，C/C++で開発されたシステムに対する SBOM 生成技術が確立できていない問題を解決するため，ビルド処理から得られる情報とその結果生成されるバイナリに含まれるメタデータに着目し，SBOM 生成手法を検討した．今後の課題として，有用な SBOM の生成にあたり必要なメタデータを得る方法を検討し，ツールの実装を目指す．

謝辞 本研究は，JSPS 科研費（JP24K14895，JP21K02862，JP23K28065，JP24H00692，JP21K18302，JP21H04877，JP23K24823，JP22K11985），2024 年度南山大学パッヘ研究奨励金 I-A-2 の助成を得て行われた．

参 考 文 献

[1] Biden, Jr., J. R.: Executive Order on Improving the Nation's Cybersecurity, `https://www.whitehouse.gov/briefing-room/presidential-actions/2021/05/12/executive-order-on-improving-the-nations-cybersecurity/`, May 2021.

[2] Collin, L.: XZ Utils backdoor, `https://tukaani.org/xz-backdoor/`, Jul 2024.

[3] Kikas, R., Gousios, G., Dumas, M., and Pfahl, D.: Structure and Evolution of Package Dependency Networks, *Proc. MSR2017*, 2017, pp. 102–112.

[4] 音田渉, 神田哲也, 眞鍋雄貴, 井上克郎, 肥後芳樹: Stack Overflow における SBOM 利活用に関する質問の分析, 信学技報, Vol. 123, No. 414, Mar 2024, pp. 127–132.

[5] The United States Department of Commerce: The Minimum Elements For a Software Bill of Materials (SBOM), `https://www.ntia.gov/report/2021/minimum-elements-software-bill-materials-sbom`, 2021.

†1 `https://github.com/curl/curl/releases/download/curl-8_10_1/curl-8.10.1.tar.xz`