実行ファイルにおける差分がある関数の特定による バージョンを変更せずに加えられた脆弱性修正の識別手法

兼松 智也[1]　山内 利宏[2,a)]　谷口 秀夫[1]

概要：IoT 機器のファームウェア（以降，FW）には，様々なアプリケーションプログラム（以降，AP）が含まれているため，AP の脆弱性管理が重要となっている．しかし，FW に対し，バージョンに依存した Software Composition Analysis ツールを利用する場合，AP のバージョンが変更されずに加えられた脆弱性修正により，修正された脆弱性を誤検知する可能性がある．本稿では，AP の実行ファイルにおいて，アップデートによりプログラムに差分が生じた関数を特定する手法を提案する．本手法により，脆弱性に関係する関数に差分があることを検出することで，脆弱性修正を検出する．具体的には，本手法では，アップデート前後の関係にある FW に含まれる実行ファイルを比較し，差分がある関数を検出する．また，差分がある関数のシンボル情報が除去され，関数名がわからない場合，シンボル情報を含めてソースコードをコンパイルした実行ファイルを使用し，関数名を特定する．評価では，シンボル情報が除去されている場合について，関数名の特定精度を明らかにする．また，実際の FW に含まれる実行ファイルを用いて，脆弱性に関係する関数が検出できることを明らかにし，本手法の有効性を示す．

キーワード：実行ファイル，比較，関数名，IoT 機器，脆弱性修正

Method for identifying vulnerability fixes added without changing the version by identifying functions with differences in executable files

Kanematsu Tomoya[1]　Toshihiro Yamauchi[2,a)]　Hideo Taniguchi[1]

Abstract: IoT device firmware (FW) contains various application programs (APs), making AP vulnerability management important. However, when using version-dependent Software Composition Analysis tools for FW, there is a risk of falsely detecting fixed vulnerabilities due to vulnerability fixes applied without a version change of the AP. This paper proposes a method for identifying functions with differences in an AP's executable file that arise from an update. This method detects vulnerability fixes by finding differences in functions related to a vulnerability. Specifically, our method compares executable files from a previous and an updated version of the firmware to detect functions with changes. If the symbol information for a changed function has been removed, making the function name unknown, we identify the function and its name using an executable file compiled from the source code with symbol information included. In our evaluation, we assess the accuracy of function name identification under conditions where symbol information has been removed. Furthermore, we demonstrate the effectiveness of our method by showing that functions related to vulnerabilities can be detected in real-world firmware executable files.

1. はじめに

近年，アプリケーションプログラム（以降，AP）の脆弱性報告数が増加している．増加する AP の脆弱性に対抗するために，ソフトウェア構成分析（Software composition

[1] 岡山大学 大学院環境生命自然科学研究科
Graduate School of Environmental, Life, Natural Science and Technology, Okayama University
[2] 岡山大学 学術研究院 環境生命自然科学学域
Faculty of Environmental, Life, Natural Science and Technology, Okayama University
[a)] yamauchi@okayama-u.ac.jp

Analysis, SCA）が注目されており，SCA のための様々なツール（以降，SCA ツール）が開発されている．SCA ツールでは，製品で利用される Open Source Software（以降，OSS）コンポーネントを分析し，脆弱性やライセンス違反などを検知することができる．また，Internet of Things（以降，IoT）機器のファームウェア（以降，FW）には様々な AP が含まれているため，SCA による脆弱性管理が重要となっている．

しかし，IoT 機器の FW に対する SCA の課題が文献 [1] で指摘されている．問題の 1 つとして，AP のバージョンを変更せずに加えられた脆弱性修正がある．これは，AP のバージョンを脆弱性が修正された最新のものにバージョンアップせずに，脆弱性を修正することである．

SCA の脆弱性検知処理が AP のバージョンに依存している場合，修正された脆弱性を誤検知してしまう可能性がある．このため，AP のバージョンを変更せずに加えられた脆弱性修正を識別する手法は重要である．

本研究では，既存の SCA ツールよりも，より正確に実行形式のバイナリファイル（以降，実行ファイル）に含まれる脆弱性を識別することを目的とする．既存の 4 つの SCA ツールで，AP のバージョンを変更せずにプログラムが変更された実行ファイルを検査したところ，AP の種別の識別に失敗するものが 3 つあった．また，AP の種別の識別ができたツールにおいて，AP のバージョンを上げずに対処された脆弱性を誤検知した．これらのことから，実行ファイルのみで脆弱性の有無を判断する手法の高度化を検討した．

本稿では，AP の実行ファイルにおいて，バージョン変更を伴わないアップデートにより，プログラムに差分が生じた関数を特定する手法を提案する．本手法により，脆弱性に関係する関数に差分があることを検出することで，脆弱性修正を検出する．

また，実際の FW に含まれる実行ファイルは，シンボル情報が除去され，関数名がわからない場合が多く，差分のある関数が特定できたとしても，関数名の特定が困難であるという課題がある．この課題について，シンボル情報を含めてコンパイルした実行ファイルと FW に含まれる AP の実行ファイルを，既存のバイナリプログラム比較ツールを用いて比較し，関数名を特定する手法を提案する．差分がある関数と脆弱性に関係する関数が同一か否か調べるためには，関数名が必要となる．しかし，FW に含まれる実行ファイルは関数名が含まれていない場合が多いため，同一プログラムのソースコードからシンボル情報を含む実行ファイルを生成し，この実行ファイルを利用することで関数名を特定する．

評価では，手法の有効性を示すために，以下の Research Question に取り組んだ．

RQ1: 既存のバイナリプログラム比較ツールはシンボル情報がない実行ファイルから関数名を特定できるか

RQ2: 本手法により，実際の FW において，バージョンを変更せずに加えられた脆弱性修正により変更があった関数を検出できるか

RQ1 に対して，シンボル情報が除去され，関数名がわからない関数に対して，既存のバイナリプログラム比較ツールにおける関数名の特定の精度を評価した．本手法による関数名の特定精度は，使用するバイナリプログラム比較ツールの精度に依存する．このため，評価の結果より，本手法における関数の特定精度を明らかにした．

RQ2 に対して，実際の FW に含まれる OSS の実行ファイルを用いて，本手法による差分がある関数の検出を試みた．これにより，脆弱性に関係する関数と関係のない関数がいくつ検出されたかを明らかにした．

2. 研究背景

2.1 SCA ツール

ソフトウェアで使用される OSS を検知し，脆弱性やライセンス違反を検知するツールとして，SCA ツールが存在する．SCA ツールでは，検査対象のソフトウェアから OSS の種別やバージョンを特定する．この情報をもとに，脆弱性データベースからソフトウェアに含まれる OSS の脆弱性を検知する．

2.2 バージョンを変更せずに加えられた脆弱性修正の適用の実態

IoT 機器の FW に含まれる AP は，脆弱性にパッチを適用することにより修正されることが文献 [2] で示されている．このような脆弱性修正では，AP のバージョンは更新されないため，バージョンに依存した SCA ツールによる脆弱性検知では，誤検知が発生する可能性がある．

また，文献 [1] では，このような脆弱性修正を加えられたと推測される実行ファイルが FW にどの程度含まれているか調査している．調査の結果，アップデート前後の関係にある FW において，同一種別 AP の実行ファイルのペア 12,946 組のうち，ハッシュ値が変化した実行ファイルは 4,960 組確認され，このうち約 44.3%がパッチによる脆弱性修正が加えられたと判断された．

このように，数多くの FW に対し，パッチによる脆弱性修正は行われていることが予想され，IoT 機器の FW の脆弱性管理の課題となることが分かる．

2.3 バージョンを変更せずに加えられたプログラム変更の特徴

コンパイルに用いる環境の違いによるプログラム変更がある場合，脆弱性修正があったとしても，ソースコードレベルの変更がバイナリプログラムのどの部分に反映されているのか差分から判断することは困難となる．文献 [3] で

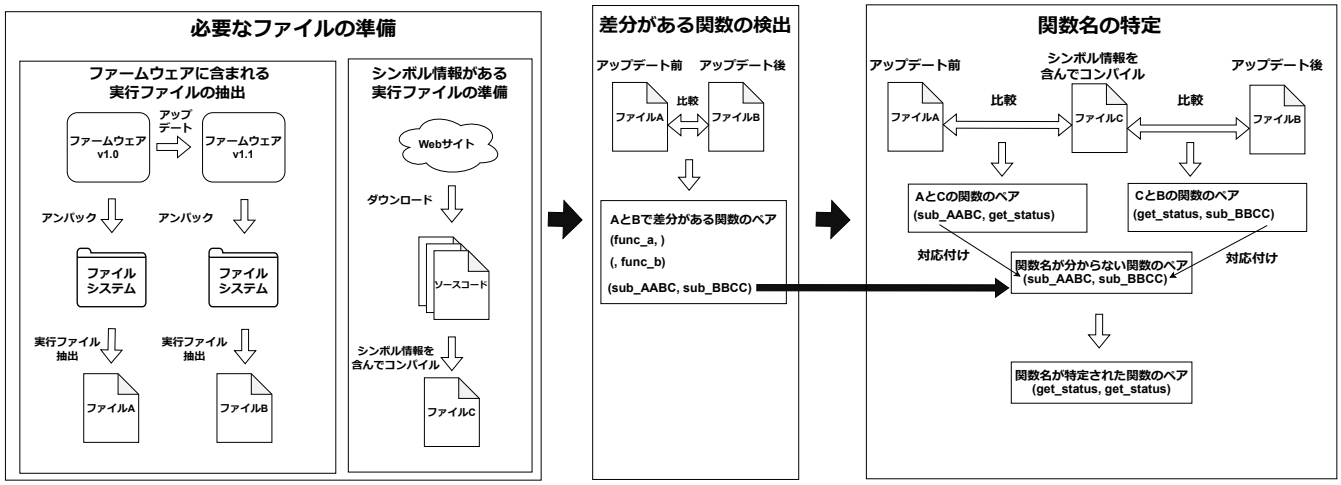

図 1 差分がある関数の特定手法

は，異なるコンパイラファミリー，コンパイラバージョン，および最適化オプションでコンパイルすると，同じソースコードから大きく異なる機械語命令列を生成する可能性があることが指摘されている．

しかし，FW に含まれる実行ファイルにおいて，バージョンを変更せずに加えられたプログラム変更は，局所的であるものが大部分を占める．文献 [4] では，11 社のベンダ製 FW591 個について，バージョンを変更せずに加えられたプログラム変更があった実行ファイル間のプログラムの類似度をバイナリプログラム比較ツール BinDiff [5] で調査している．調査の結果，バージョンを変更せずに加えられたプログラム変更があった実行ファイルのうち，94.5% が類似度 0.885 を超える小さな変更であることが明らかとなった．なお，類似度は，0 から 1 の値であり，1 に近いほどプログラムが似ていることを表す．

プログラムが局所的に変更される場合，差分がある関数を検出することで，プログラムの変更がどのような理由によるものか推測できる可能性がある．このため，差分がある関数を特定し，脆弱性に関係する関数に変更があることが分かれば，バージョンを変更せずに加えられる脆弱性修正を識別できると推察できる．

3. 差分がある関数の特定手法

3.1 目的

本手法の目的は，実行ファイルに対し，アップデートにより生じた差分がある関数を特定することである．差分がある関数を特定することにより，脆弱性に関係する関数に差分があることが分かれば，アップデート後の FW に含まれる実行ファイルに対し，バージョンを変更せずに加えられた脆弱性修正を識別することができる可能性がある．

3.2 差分がある関数の特定の課題と対処

3.2.1 課題

差分がある関数を特定するうえで，FW に含まれる実行ファイルのシンボル情報が除去されている場合，関数名が特定できないという課題がある．差分がある関数のアドレスと処理を特定したとしても，関数名が分からなければ，どの関数に変更が加えられたか特定することは困難となる．このため，脆弱性に関係する関数か否かを判断するために，差分がある関数の関数名を特定することは重要となる．

3.2.2 課題に対する対処

課題に対する対処として，関数名が分からない関数に対し，シンボル情報が存在する実行ファイルの関数と比較し，元々同一の関数名を持っていたと推測できる関数のペアを作ることで，関数名を特定する．また，シンボル情報が存在する実行ファイルを比較に用いるために，AP の公式サイトからソースコードをダウンロードし，シンボル情報を含めてコンパイルし，実行ファイルを生成することを想定している．

3.3 差分がある関数の特定手順

3.3.1 手法の概要

本手法では，アップデート前後の関係にある FW から同一種別かつ同一バージョンの AP の実行ファイルを抽出する．また，FW に含まれる AP と近いバージョンの AP から，シンボル情報を含まれた実行ファイルを取得する．次に，これらのファイル間で差分がある関数を検出する．最後に，検出した差分がある関数に関数名が存在しない場合，FW に含まれていた実行ファイルとシンボル情報が含まれている実行ファイルを比較し，検出した関数の関数名を特定する．

詳細を**図 1** に示す．なお，実行ファイルについて，アップデート前の FW に含まれるものを A，アップデート後の

FW に含まれるものを B と表す．また，シンボル情報を含めてソースコードをコンパイルして生成した実行ファイルを C と表す．以降では，各手順の説明を行う．

3.3.2 必要なファイルの準備

FW に含まれる実行ファイルの抽出：アップデート前後の関係にある FW について，アンパックして抽出したファイルシステムから，同一種別で同一バージョンの AP の実行ファイルを取り出す．（ファイル A, B）

シンボル情報がある実行ファイルの準備：対象としている AP と同一かつ近いバージョンの AP のソースコードを Web サイトからダウンロードし，シンボル情報を含めてコンパイルし，実行ファイルを取得する．（ファイル C）

3.3.3 差分がある関数の検出

抽出した FW に含まれる実行ファイル A と B を BinDiff で比較し，差分がある関数を検出する．本研究では，以下の 3 種類について，差分がある関数とする．

- アップデート前の FW に含まれる実行ファイルにのみ存在
- アップデート後の FW に含まれる実行ファイルにのみ存在
- 両方に存在し，処理が同一でない

処理が同一でないという判定は，BinDiff の類似度スコアである Similarity に基づいて行う．具体的には，関数の Similarity が 1.0 未満である場合，処理が同一でないと判定する．関数の Similarity は，関数の制御フローグラフや命令レベルの構造を比較し，複数の要素に基づいて加重平均で算出される．具体的には，制御フローグラフのエッジの一致率 (25%)，基本ブロックの一致率 (15%)，命令の一致率 (10%)，および制御フローグラフの MD-index 差分 (50%) の加重平均である．MD-index とは，グラフ構造の特徴を数値に変換するハッシュアルゴリズムである．

また，関数が外部関数呼び出しのためのコードである場合，差分がある関数から除外する．BinDiff では，実行ファイル内の `.plt` セクションにあるような外部関数を呼び出すためのコードについて，関数とみなし，差分を調査する．外部関数を呼び出すためのコードに変更があったとしても，プログラムの処理の差分とはいえない．このため，本研究における差分がある関数から，外部関数呼び出しのためのコードを除外する．

3.3.4 関数名の特定

差分がある関数の検出の手順において検出された関数名がわからない関数に対し，シンボル情報がある実行ファイル内の関数をもとに関数名を特定する．具体的には，バイナリプログラム比較ツールにより，シンボル情報があるファイル C の関数とファイル A と B に含まれる関数名がわからない関数を比較し，一致する関数を対応付けすることで関数名を特定する．なお，本研究では，バイナリプログラム比較ツールにより一意に関数のマッチングが成功している場合，関数が一致していると判断する．

4. 評価

4.1 評価目的と実施項目

評価は，本手法による差分がある関数の特定の有効性を示すことを目的とする．以下の 2 つの Research Question について取り組んだ．

RQ1: 既存のバイナリプログラム比較ツールはシンボル情報がない実行ファイルから関数名を特定できるか

RQ2: 本手法により，実際の FW において，バージョンを変更せずに加えられた脆弱性修正により変更があった関数を検出できるか

RQ1 では，既存のバイナリプログラム比較ツールにより，シンボル情報が除去された実行ファイルにおける関数名が分からない関数に対して，どの程度関数名を特定できるか否かを評価する．これにより，本手法において，関数の特定がどの程度できるのか明らかにする．シンボル情報を含めてコンパイルした実行ファイルとシンボル情報を除去した実行ファイルとの比較で，関数名が分からない関数の特定精度を評価した．

関数の特定精度は，関数の一致の判定に用いるバイナリプログラム比較ツールを変えて評価する．また，比較する実行ファイルの生成時に使用するコンパイラのバージョンと最適化オプションを変更して評価した．ソースコードからコンパイルする環境を，特定の対象である実行ファイルが含まれる FW のコンパイル環境に合わせるのは困難である．このため，関数名の特定において，コンパイル環境が違う場合における精度が重要となる．つまり，コンパイル環境の違いが与える影響が，関数名の特定に影響を与えないことが重要である．

RQ2 では，FW に含まれており，バージョンを変更せずに加えられた脆弱性修正によるプログラム変更があった実行ファイルについて，3 章で説明した手法により，差分がある関数を検出した．この結果から，脆弱性に関係する関数を検出できるか否か明らかにする．また，本手法により，バージョンを変更せずに加えられる脆弱性修正を検出できるか否かを明らかにする．

4.2 既存ツールによる関数名の特定精度の評価

4.2.1 評価方法

本評価では，シンボル情報を含めてソースコードをコンパイルして生成した実行ファイルとシンボル情報が除去された実行ファイルを用いて，既存のバイナリプログラム比較ツールの関数の特定精度を評価した．比較対象として，シンボル情報を含む実行ファイルと `strip` コマンドでシンボル情報が除去された実行ファイルを用いた．具体的には，シンボル情報を含む実行ファイル 1 つに対し，コンパイラのバージョン，および最適化オプションを変更してコ

ンパイルし，シンボル情報を除去した複数の実行ファイルを比較した．

まず，用意したファイルを比較し，一致する関数のペアを作成した．次に，シンボル情報が除去された実行ファイルの中で，関数名が分からないものについて，正しく関数名を特定できた関数の数を集計し，この割合を求めた．これにより，コンパイルする環境が異なる状況において，関数名の特定がどの程度可能かを明らかにした．

評価に用いるバイナリプログラム比較ツールは以下の 2 種類とした．

(1) BinDiff

(2) Diaphora

BinDiff と Diaphora [6] はバイナリプログラム比較ツールの代表的なツールであるため用いた．BinDiff は制御フローグラフを比較し，関数のマッチングを行う．Diaphora は制御フローグラフに加えて，逆コンパイル結果の比較が行えるツールである．Diaphora は BinDiff より多くの特徴をバイナリプログラムの比較に用いることができ，逆コンパイル結果の比較はこのうちの 1 つである．なお，比較に用いる逆コンパイル結果は，IDA Pro [7] の逆コンパイラが生成したものを使用した．

4.2.2 評価対象の実行ファイルの選定

関数名特定の精度を評価するために用いる実行ファイルについての詳細を**表 1** に示す．本評価では, Ubuntu 24.04.1 TLS 上で Buildroot-2025.05.x を用いて wpa_supplicant のバージョン 2.11 をコンパイルして生成した実行ファイルを用いた．また，GCC 13.3.0 と最適化オプション`-O2` を用いてコンパイルし，シンボル情報を含んだ実行ファイルを基準とした．これに対し，コンパイラのバージョンや最適化オプションを変更してコンパイルし，シンボル情報を除去した複数の実行ファイルを比較対象とした．具体的には，コンパイルする環境に関して，基準としたシンボル情報を含む実行ファイルから，最適化オプションを`-O0`，`-O1`，`-O3` に変更したファイル，およびコンパイラバージョンを GCC 12.4.0 に変更したファイルを用意した．

表 1 関数名の特定対象の実行ファイル

ファイル種別	コンパイラ	最適化オプション
シンボル情報有のファイル	GCC 13.3.0	`-O2`
比較対象実行ファイル A	GCC 13.3.0	`-O0`
比較対象実行ファイル B	GCC 13.3.0	`-O1`
比較対象実行ファイル C	GCC 13.3.0	`-O3`
比較対象実行ファイル D	GCC 12.4.0	`-O2`

4.2.3 評価結果

評価結果について，**表 2** に示す．BinDiff による比較では，指定した最適化オプションを同じにして，GCC のバージョンを変更した実行ファイルに対し，98.0%と高い精度で特定できた．また，GCC のバージョンを同じものにして，コンパイル時に指定した最適化オプションを`-O3` にした実行ファイルに対し，83.3%と高い精度で特定できた．一方で，GCC のバージョンを同じものにして，指定した最適化オプションを`-O0`，`-O1` にした実行ファイルに対し，それぞれ 6.4%, 49.6%と低い精度となった．結果から，GCC 13.3.0 と GCC 12.4.0 では，BinDiff による関数名の特定に影響を大きく与えるほど，生成される機械語命令列に差異がない可能性がある．また，最適化オプションが違う実行ファイルに対し，関数名を特定することが難しいことが分かる．

Diaphora による比較では, 最適化オプションが`-O0`，`-O1` の実行ファイルに対し，それぞれ 25.0%, 74.7%と BinDiff による比較よりも高い精度で特定できた．一方で，GCC のバージョンを変更した実行ファイル，およびコンパイル時に指定した最適化オプションが`-O3` である実行ファイルに対し，高い精度ではあるものの，BinDiff より精度が低くなった．結果から，Diaphora では，コンパイル時に指定した最適化オプションが違う実行ファイルの比較に対し，精度よく対応付けが可能である可能性がある．

結果から，**RQ1** に対し，以下のことが分かる．コンパイル時に使用したコンパイラのバージョンが違う実行ファイルに対し，十分な精度で関数名を特定できた．しかし，コンパイル時に指定した最適化オプションによっては特定精度に課題がある．

4.3 差分がある関数の検出の評価

4.3.1 評価方法

本手法を実際の FW に含まれる実行ファイルに対し，適用する．検出した差分がある関数を示し，脆弱性に関係する関数，および無関係な関数がどれだけ含まれているかを示す．対象とする実行ファイルの AP として wpa_supplicant，対象とする脆弱性として Key Reinstallation Attacks (KRACKs) 脆弱性を選定した．wpa_supplicant は，文献 [2] で，メーカ側が修正対応済みとしているものに対し，SCA ツールで KRACKs 脆弱性が検知されたことが報告されている．この事例から，評価対象として適切であると判断し，wpa_supplicant を選択し

表 2 関数名の特定精度評価結果

バイナリ比較ツール名	比較対象実行ファイル			
	A	B	C	D
BinDiff	143/2230 (6.4%)	674/1305 (49.7%)	1021/1226 (83.3%)	1255/1281 (98.0%)
Diaphora	558/2230 (25.0%)	975/1305 (74.7%)	984/1226 (80.3%)	1147/1281 (89.5%)

表 3 差分がある関数の検出の評価における対象の実行ファイル

	シンボル情報を含めてコンパイルした実行ファイル	FW に含まれる実行ファイル
AP 名	wpa_supplicant	wpa_supplicant
バージョン	2.0	2.0-devel
アーキテクチャ	ARM (32bit)	ARM (32bit)
コンパイラ	GCC 4.7.3	GCC 4.3.3
最適化オプション	-O2	不明

た．なお，KRACKs 脆弱性に関係する関数か否かの判断は，Web サイト [8] のパッチ情報から行った．

4.3.2 評価対象の実行ファイルの選定

表 3 に評価に用いる実行ファイルについての詳細を示す．シンボル情報を含めてコンパイルした実行ファイルのバージョンは，FW に含まれる実行ファイルのバージョン 2.0-devel に近いと考えられるバージョン 2.0 のものを用いた．また，シンボル情報を含めてコンパイルした実行ファイルについて，Buildroot-2013.08.1 を用いて生成した．

実行ファイルを抽出する FW は，SCA ツールで脆弱性を誤検知したものを選択する．商用の SCA ツールを用いて，KRACKs 脆弱性の誤検知を確認した．

4.3.3 評価結果

評価結果について，差分があった関数を本手法で検出した結果を**表 4** に示す．脆弱性に関係する関数が 3 つ，脆弱性に関係のないその他の関数が 5 つ検出された．その他の関数が検出された原因については，5.3 節で考察する．

RQ2 について，以下のことが分かる．脆弱性に関係する関数が検出できた．一方で，脆弱性に関係ない関数にも差分が検出された．

5. 考察

5.1 指定した最適化オプションによる生成される機械語命令列の違い

4.2.3 項における関数名の特定精度の評価結果において，基準となる-O2 でコンパイルした実行ファイルと-O0 や-O1 でコンパイルした実行ファイルとの比較では，あまり精度が良くなかった．一方で，-O2 と-O3 の比較では，評価で試した 2 つのバイナリプログラム比較ツールで 80%を超える高い精度であった．この違いについて，-O2 は-O3 と生成される機械語命令列が近いという可能性が考えられる．バイナリプログラムから最適化オプションを推定する手法を提案した文献 [9] では，-O2 と-O3 の誤識別が多いことが示されている．これは，生成される機械語命令列が近いためであると考えられる．このため，-O2 と-O3 では，生成された機械語命令列が近く，関数の一致を探索することが容易である可能性がある．

5.2 関数名特定のための改善点

4.2 節の結果から，コンパイラのバージョンや最適化レベルが違うことで，関数名の特定精度が変わることが分かった．このため，コンパイラのバージョンや最適化レベルによって変更しにくい特徴に重きをおいて，一致する関数のペアを作成する必要がある．具体的には，参照している文字列，呼び出し関係に重きを置いた比較が有効であると推察する．

5.3 脆弱性修正に関係のない関数の検出の原因

差分がある関数がインライン展開されている場合，関数を呼び出している他の関数も差分があると判定される可能性がある．これは，ソースコード自体に変更がない関数であっても，呼び出し先の関数がコンパイル時にインライン展開されることで，呼び出し元関数の処理に差分が生じるためである．このような場合，インライン展開された関数の本体が実行ファイル上に残っていれば，呼び出し元の関数の処理とインライン展開された関数処理の部分一致を見つけることで，呼び出し元の関数の誤検出を削減できる可能性がある．

また，BinDiff による差分の誤検出が考えられる．差分がある関数のうち，2 つは，逆コンパイル結果に差分がなかった．このため，関数の差分の判断基準として，逆コンパイル結果が同一であるか否かを用いることで，脆弱性に関係のない関数を除外することができると推察する．

6. 関連研究

IoT 機器を対象とした SCA の研究として文献 [1], [2] がある．文献 [1] では，IoT 機器に対する SCA の課題を明らかにするために，バージョンを変更せずに加えられた脆弱性修正がどの程度含まれているのか調査した結果を報告している．文献 [2] では，IoT 機器のうち，ルータの FW を対象に，SCA ツールを用いて，メーカ側が修正対応済みとしている脆弱性が検知された事例を報告している．本研究

表 4 ファームウェアに含まれる実行ファイルに対する差分がある関数の検出結果

脆弱性に関係する関数	その他の関数
wpa_sm_drop_sa	wpa_supplicant_rx_eapol
wpa_sm_notify_assoc	wpa_config_write
wpa_supplicant_install_gtk	wpa_sm_rx_eapol
	wpa_supplicant_rx_eapol_bridge
	wpa_supplicant_ctrl_iface_drop_sa

では，このような脆弱性修正によって引き起こされる可能性がある脆弱性検知における誤検知に対処するための手法を検討している．

シンボル情報が除去されたバイナリプログラムから関数名を推測する関連研究として文献 [10], [11] がある．文献 [10] では，関数の呼び出し関係や実行時の振る舞いを考慮して学習したニューラルネットワークを用いて関数名を特定する手法を提案している．文献 [11] では，LLM を活用し，訓練データへの過学習によって未知のバイナリプログラムに対して汎化性能が低くなるという既存の機械学習による手法の課題を克服した手法を提案している．本研究では，同じ AP のソースコードが入手可能である前提で，既存のバイナリプログラム比較ツールを用いた関数名の特定を行っている．

バイナリプログラムを対象としたパッチによる脆弱性修正の識別の研究として文献 [12], [13] がある．文献 [12] では，ディープラーニングに基づく静的解析で脆弱性の候補を絞り込み，動的解析を用いて誤検知を大幅に削減することで，既知脆弱性を検知し，パッチが適用されているか判断する手法を提案している．文献 [13] では，脆弱な関数とパッチ適用済み関数から特徴を抽出し，この特徴との類似度でパッチが適用されているか判断する静的解析の手法を提案している．本研究では，アップデート間のプログラムの差分が小さいという知見をもとに，差分があった関数を検出することで脆弱性修正があったことを推測する．

バイナリプログラムの比較に関する研究として，静的解析，動的解析，および機械学習によるバイナリプログラムの類似性を検出する研究がある．静的解析では，BinDiff の比較手法の元となった文献 [14] により，関数のコールグラフと制御フローグラフを用いて，関数や基本ブロックの対応付けを行う手法を提案されている．動的解析では，文献 [15] により，シンボリック実行とシステムコールを用いて難読化されたプログラムを比較できる手法が提案されている．機械学習では，文献 [16] により，機械学習を用いて命令列やグラフの特徴を埋め込み表現として学習し，関数同士の類似性を検出する手法が提案されている．本研究では，既存の代表的なバイナリプログラム比較ツールを用いており，比較は静的解析により行っている．

7. おわりに

バージョンを変更せずに加えられた脆弱性修正識別のために，FW のアップデートにより生じた差分がある関数を実行ファイルから検出し，関数名を特定する手法を提案した．手法の有効性を評価するために，既存のバイナリプログラム比較ツールを用いて，関数名の特定精度を明らかにした．また，実際の FW に含まれる実行ファイルを用いて，脆弱性に関係する関数を検出できるか否か試みた．

結果として，コンパイル時に使用したコンパイラのバージョンが違う実行ファイルに対し，最大 98.0%と十分な精度で関数名を特定できることが分かった．一方で，コンパイル時に指定する最適化オプションが違う場合，特に最適化オプション`-O2` と`-O0` を指定した実行ファイルの比較では，約 25.0%の精度でしか特定することができなかった．また，実際の FW に対して，本手法により，脆弱性に関係する関数が検出できたものの，脆弱性に無関係な関数も検出された．さらに，関数の誤検出の要因を分析し，誤検出を減らすための対策についても述べた．

今後の課題として，違うコンパイラのバージョンや最適化レベルで生成されたバイナリプログラムに対し，関数名の特定精度の向上がある．

謝辞　本研究の一部は，JST【経済安全保障重要技術育成プログラム】【JPMJKP24K2】の支援を受けたものです．

参考文献

[1] Akiyama, M., Shiraishi, S., Fukumoto, A., Yoshimoto, R., Shioji, E. and Yamauchi T.：Seeing is not always believing: Insights on IoT manufacturing from firmware composition analysis and vendor survey, *Computers & Security,* vol. 133, p. 103389 (2023).

[2] 木原百々香，佐々木貴之，吉岡克成：ルータのファームウェアに含まれる OSS 脆弱性に関する SCA ツールを用いた調査，電子情報通信学会技術研究報告, vol. 124, no. 124, pp. 58–65, (2024).

[3] Du, Y., Alrawi, O., Snow, K., Antonakakis, M. and Monrose, F.: Improving Security Tasks Using Compiler Provenance Information Recovered At the Binary-Level, *In Proc. of the 2023 ACM SIGSAC Conference on Computer and Communications Security,* pp.2695—2709, (2023).

[4] 兼松智也，山内利宏：IoT 機器のファームウェアにおけるハードウェアバージョンを考慮したソフトウェアの差分解析，第 24 回情報科学技術フォーラム (FIT2025), (2025).

[5] Google: BinDiff, 入手先〈https://github.com/google/bindiff〉(参照 2025-08-21).

[6] J. Esparza: Diaphora, 入手先〈https://github.com/joxeankoret/diaphora〉(参照 2025-08-21).

[7] Hex rays: IDA Pro, 入手先〈https://hex-rays.com/ida-pro/〉(参照 2025-08-21).

[8] hostapd and wpa_supplicant: Index of /security/2017-1, 入手先〈https://w1.fi/security/2017-1/〉(参照 2025-08-21).

[9] Pizzolotto, D. and Inoue, K.: Identifying Compiler and Optimization Level in Binary Code From Multiple Architectures, *in IEEE Access,* vol. 9, pp. 163461–163475, (2021).

[10] Jin, X., Pei, K., Yeon, Won, J.Y. and Lin, Z.: SymLM: Predicting Function Names in Stripped Binaries via Context-Sensitive Execution-Aware Code Embeddings, *Proc. of the 2022 ACM SIGSAC Conference on Computer and Communications Security,* pp. 1631—1645 (2022).

[11] Jiang, L., Jin, X. and Lin, Z.: Beyond Classification: Inferring Function Names in Stripped Binaries via Domain Adapted LLMs, *Proc. 2025 Network and Distributed System Security Symposium,* (2025).

[12] Sun, P., Garcia, L., Salles-Loustau, G. and Zonouz, S.:

Hybrid Firmware Analysis for Known Mobile and IoT Security Vulnerabilities, *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*(DSN), pp. 373–384, (2020).

[13] Xu,Y., Xu, Z., Chen, B., Song, F., Liu, Y. and Liu, T.: Patch based vulnerability matching for binary programs, *Proc. of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis,* pp. 376–387, (2020).

[14] Dullein, T. and Rolles, R.: Graph-based comparison of Executable Objects (English Version), *Sstic,* vol. 5, no.1, p. 3 (2005).

[15] Ming, J., Xu, D. and Jiang, Y. and Wu, D.: BinSim: Trace-based semantic binary diffing via system call sliced segment equivalence checking, *In 26th USENIX Security Symposium* (USENIX Security 17), pp. 253–270, (2017).

[16] Duan, Y., Li, X., Wang, J. and Yin, H.: DeepBinDiff: Learning program-wide code representations for binary diffing, *In Network and Distributed System Security Symposium,* (2020).