バイナリファジングを用いた正常なプログラムとの挙動比較によるネットワークIoT機器のトロイ化検出手法の評価

金城 豪志[1]　高田 雄太[2]　熊谷 裕志[2]　神薗 雅紀[2]　山内 利宏[3]

概要：IoT 機器の普及に伴い，IoT 機器を標的とした攻撃が増加している．特に外部ネットワークやサプライチェーン攻撃を通じて不正なコードが組み込まれるトロイ化は検出が困難である．過去に我々は，正常な IoT 機器と検査対象 IoT 機器に同一のテストデータを用いたバイナリファジングを行い，得られたシステムコールの発行順序を比較することで，トロイ化を検出する手法を提案した．本研究では本手法のさらなる実践的な有用性を検証するため，バックドアデータセット ROSARUM のうち，実在するトロイ化バイナリを含む authentic を用いた評価を行う．提案手法で使用する AFL++によるファジングは標準入力を使用するため，環境変数の設定や共有ライブラリの使用に加え，`strace` のオプション設定により authentic に含まれるソケット通信が必要なバイナリにも対応できるよう本手法を拡張した．その結果，提案手法は実在するトロイ化も検出できるとともに，既存手法 ROSA と比較して誤検出を抑制できることを示す．

Evaluation of a Trojan Detection Method for IoT Devices Using Binary Fuzzing and Behavioral Comparison with a Known-Clean Program

Tsuyoshi Kaneshiro[1]　Yuta Takata[2]　Hiroshi Kumagai[2]　Masaki Kamizono[2]
Toshihiro Yamauchi[3]

Abstract: With the widespread adoption of IoT devices, attacks targeting them have been increasing. One particularly difficult to detect threat is Trojanization, in which malicious code is embedded through external networks or supply chain attacks. In our previous work, we proposed a method to detect such Trojanization by performing binary fuzzing on both a known clean IoT device and a target IoT device using identical test data, and then comparing the sequences of issued system calls. In this study, to further evaluate the practical effectiveness of the method, we conducted experiments using the "authentic" subset of the ROSARUM backdoor dataset, which contains real-world Trojanized binaries. Since fuzzing with AFL++ in our method uses standard input, we extended the approach to support binaries in "authentic" that require socket communication by configuring environment variables, shared libraries, and `strace` options. Experimental results show that the proposed method can detect real-world Trojanization and, compared to the existing method ROSA, can reduce false positives.

1. はじめに

Internet of Things（以降，IoT）機器の普及に伴い，IoT 機器を標的としたサイバー攻撃が増加している．この攻撃の中には，外部ネットワーク経由でのマルウェア感染 [1] や，製造や流通過程において不正なコードが混入されるサプライチェーン攻撃 [2] がある．特に，特定の条件（以降，トロイトリガー）のみ悪意のある振る舞い（以降，トロイペイロード）をするトロイの木馬型マルウェアや，バックドアに関連する不正なコードが組み込まれた IoT 機器は，これらの機能の検出困難さが問題となる．これは，トロイトリガーが攻撃者しか知り得ない条件になっているため

[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, Okayama, 700–8530 Japan
[2] デロイト トーマツ サイバー合同会社
Deloitte Tohmatsu Cyber LLC, Tokyo, 100–0005 Japan
[3] 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, Okayama, 700–8530 Japan

である．トロイ化を検出するために，IoT 機器を構成するファームウェアやバイナリファイルを検査する必要があるものの，ソフトウェア規模が大きく，検査対象の件数が多い場合，手作業による検査は現実的ではない．

そこで，本研究では，トロイ化の検出に有効な手法としてファジングに着目した．ファジングは，検査対象プログラム（Program Under Test, 以降 PUT）に入力されるデータ（以降，テストデータ）を自動生成し，PUT にテストデータを入力することにより，予期せぬ挙動を効果的かつ効率的に引き出し，未知の脆弱性や不具合を検出する．

バイナリファジングによりバックドアを検出する先行研究として，ROSA [3] がある．ROSA は，類似した入力は，類似した挙動を示すという考えに基づき，2 つのフェーズにおけるファジングで生成された入力で類似したペアの発行されたシステムコールの種類を比較し，この差分からバックドアを検出する．しかし，ROSA は入力間の類似性をヒューリスティックな推定に依存するため，過去の我々の検証では，わずかな実行経路の違いによるシステムコールの差異による誤検出や，誤検出に伴う調査工数の増大といった課題が明らかとなった [4].

そこで，我々は，検査対象 IoT 機器と同型の正常な IoT 機器を用意し，各機器から抽出したプログラムに対するバイナリファジング中に取得したシステムコールを比較することによって，トロイ化を検出する手法を提案した [4]. 本手法は同じ条件下で適用したファジングにおいて，発行されたシステムコールの実行順序を比較し，得られた差分から，トロイ化による悪意のある動作を検出できる．この正常なプログラムとの挙動比較により推定に依存しないため，ROSA の課題を解決できる．これはライブラリ等の様々な種類のプログラムを含む ROSARUM の合成バックドア（synthetic）データセットを用いた評価によって示し，提案手法の有用性を示した．しかし，ROSARUM の実在するトロイ化バイナリを含むデータセット（authentic）に対する有用性は未検証であった．

そこで，本稿では，authentic を用いて，提案手法が実際の攻撃事例でも有効か否かの実践的有用性を検証する．評価を行う上で，authentic は，ソケット通信を行う httpd や vsFTPd 等のネットワーク IoT 機器で動作するプログラムを含むため，標準入力で動作するファザー AFL++でも提案手法を適用できるよう，環境変数の設定や共有ライブラリの使用を通じ提案手法を拡張した．また，トロイ化関連のシステムコールに絞り，文献 [4] より誤検知を削減した．

2. 研究背景

2.1 IoT 機器のトロイ化

IoT 機器のトロイ化とは，IoT 機器の利用者に悪意のある振る舞いをしていることを認識させずに，情報窃取や外部との通信などを行うことである．悪意のあるコードが組み込まれる主なケースとしては，以下の 2 つがある．

(1) **外部からネットワーク経由で，悪意のあるコードが利用中の IoT 機器に組み込まれるケース [5]：**攻撃者はネットワーク経由で IoT 機器の内部に侵入し，正常なソフトウェアをこれと同じ悪意のあるコードが組み込まれたソフトウェアに差し替えることによって，通常の動作を装いながら，IoT 機器の内部情報を抜き取られる可能性がある．

(2) **サプライチェーン経由でファームウェアに悪意のあるコードが組み込まれるケース [6], [7]：**これは，ソフトウェアサプライチェーン攻撃と呼ばれ，製造工程で，ソフトウェアにバックドアのような不正なコードを混入させる．また，攻撃者しか知り得ない複雑な条件で起動するよう隠されているため，利用者に認識されず，権限昇格，および基盤となるシステムリソースに不正アクセスを行い，IoT 機器の内部情報を抜き取られる可能性がある．実際，PHP, ProFTPD, vsFTPd, および xz などのオープンソースプロジェクトにバックドアが組み込まれた事例がある [8], [9].

2.2 ファジング

ファジングは，自動生成した大量のテストデータをソフトウェアに入力し，未知の脆弱性や不具合を検出するテスト手法である [10], [11]．テストデータは主に遺伝的アルゴリズムに基づいて生成され，検査対象が想定していない実行経路を探索することで，脆弱性や不具合を検出する．

実行経路の探索方法に応じて，ファジングは，ホワイトボックス，ブラックボックス，およびグレイボックスに大別できる [12]．本研究で利用するグレイボックスファジングは，軽量な静的解析や動的実行トレースを用いてプログラムのコードカバレッジが上昇するようにテストデータを生成する．このため，ホワイトボックスより効率的で，ブラックボックスより探索範囲が広い特徴がある [13], [14], [15].

また，検査対象によっても分類され，本研究で利用するバイナリファジングは，バイナリ形式のプログラムを対象に，テストデータを入力してメモリやレジスタの値を観測することで，バグ等を検出する．この手法は，プログラム内部の情報にアクセスしやすいため，トリガー特定が容易な反面，デバイス上でファジング処理のリソースを要する．

2.3 先行研究：ROSA

2.3.1 概要

Kokkonis らは，グレイボックスファジングを応用してコードレベルのバックドアの自動検出手法 ROSA [3] を提案している．コードレベルのバックドアは，外部からは隠された状態でプログラム内に組み込まれた機能であり，特定の入力により，認証バイパスや権限昇格などの不正な動作を引き起こす [16].

ROSA は，AFL++ [15] を基盤とし，新たに設計されたメタモルフィックテストオラクル [17] を組み合わせること

で，バックドアを検出する．メタモルフィックテストオラクルとは，データは異なるが本質的には同じ意味を持つ入力のペア（以降，入力ファミリ）に対して，出力が一致または類似することを期待するテスト手法である．ROSA は，同じ入力ファミリ内の入力であれば，そのシステムコールも一致または類似すべきであるというメタモルフィック関係を仮定する．したがって，同じ入力ファミリに属する 2 つの入力によるシステムコールに差が生じた場合，ROSA はバックドアが動作した可能性があるとして検出する．

ROSA のバックドア検出報告は，2 つのフェーズからなる．フェーズ 1 の目的は，入力ファミリごとの代表入力を集めることである．AFL++で PUT に対して短時間のファジングを適用し，異なる制御フローグラフ（以降，CFG）エッジの組み合わせを通る入力を代表入力として収集する．

フェーズ 2 の目的は，PUT に対して長時間のファジングを行い，バックドアを検出することである．ファジングで生成されるテストデータごとに，フェーズ 1 で収集した代表入力の中から最もハミング距離の近い CFG エッジパターンを持つ入力を選定する．選定した代表入力と生成されるテストデータに対して，プログラムが発行するシステムコールの種類を比較する．比較結果に差が発生した場合，バックドアの検出を報告する．この検出報告を受け，最終的に専門家が詳細な調査を行い，バックドアを検出する．

2.3.2 先行研究の課題

ROSA は一定数のトロイ化を検出できるものの，我々の検証により，ROSA の 2 つの課題を指摘した [4].

1 つ目の課題は，トロイ化の誤検出を引き起こすことである．これは，ROSA の類似性の判定が CFG エッジの近似情報に依存し，微細な実行経路の違いからシステムコールの差が発生する可能性があるためである．我々の検証で，正常なプログラムからバックドアを報告されるケースを確認した．これは，調査工数を増大させる要因となる．

2 つ目の課題は，単純なトロイトリガーの見逃しである．ROSA はフェーズ 1 で収集した代表入力に基づき，フェーズ 2 でテストデータを生成し比較する．しかし，代表入力が既にバックドアの条件を満たしていた場合，比較対象の双方で同じ悪意ある挙動が発生し，差分がなく見逃される可能性がある．この現象はバックドア汚染 [3] と呼ばれ，特にトリガー条件が単純な場合に顕著である．我々の検証で，ファジングで引き当てやすい条件である文字列長比較の条件を持つ libxml2 のバックドアで検証したところ，ROSA のバックドア汚染による多くの見逃しを確認した．

3. バイナリファジングを用いた正常なプログラムとの挙動比較による IoT 機器のトロイ化検出手法

文献 [4] において，IoT 機器および機器内のバイナリ形式のプログラムにアクセスできる前提の下，ROSA の課題を解決する手法を提案した．本章では，本手法の設計方針や処理プロセスを概説するとともに，残課題を整理する．

3.1 設計方針

ROSA の課題に対処する設計方針を説明する．本研究で用いる提案手法は，検査対象 IoT 機器に含まれるプログラム PUT，および同型の正常な IoT 機器に含まれるプログラムの挙動を比較することで，推定に依存せず，明確な基準に基づきトロイ化を検出する．このアプローチにより，ROSA の課題を解決する．ここで，検査対象 IoT 機器とは，悪意のあるコードが組み込まれた可能性のあるプログラムを含む機器とする．正常な IoT 機器とは，検査対象のトロイ化判別の基準とするため，悪意のあるコードが組み込まれていないプログラムで構成される同型の機器とする．ファジングで生成された全てのテストデータで両プログラムを実行し，システムコールを比較する実装では，計算コストが大幅に増大してしまうため，以下に示すファジングとシステムコール比較の 2 段階の手順を採用した．

第 1 段階では，トロイトリガーを発見することを目的とし，ファジングを PUT に対してのみ行う．この段階では，ファジングによって探索された実行経路を表す新たに分岐を通過したテストデータを取得する．

第 2 段階では，第 1 段階で新たに分岐を通過したテストデータを PUT と正常なプログラムの両方で改めて実行する．これにより，ファジングによって発見された実行経路を正常なプログラム上でも正確に再現し，この上で，両プログラムのシステムコール情報を収集・比較する．この設計により，PUT のみへのファジングとファジング中のシステムコール収集を省略可能にしたこと，システムコールの収集回数の削減，および正常な挙動との比較による高い検出精度と調査工数の削減という利点を実現した．

この正常な IoT 機器を用いることにより，推定に依存せず，明確な基準に基づく比較が可能になる．これにより，ROSA の 1 つ目の課題である誤検出を削減できるとともに，トロイトリガーが単純であった場合でも，見逃しを防ぎ，ROSA の 2 つ目の課題である単純なトロイトリガーによるトロイ化の見逃しに対処できる．

3.2 処理プロセス

前節の方針に基づき設計した提案手法の全体像を**図 1** に示す．図 1 より，提案手法のプロセスは，(1) ファジング処理，(2) システムコールの比較の 2 つの手順で構成される．

3.2.1 手順 1 ファジング処理

手順 1 では，検査対象 IoT 機器に含まれるプログラムに対して，グレイボックスファザーの AFL++ [15] を用いたファジングを適用する．AFL++は，ファジング中に新たに分岐を通過したテストデータを保存し，後述するシステムコールの比較に利用する．なお，本研究におけるファジ

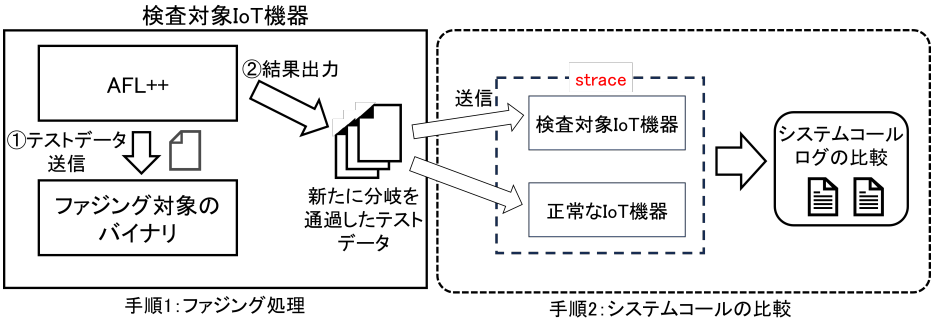

図 1 提案手法の全体像

ング対象は IoT 機器に組み込まれたバイナリ形式のプログラムを想定し，QEMU モードの AFL++を利用した．また，提案手法では，複数プロセスによる効率的なファジングのために，`-S` および`-M` オプションを利用した．さらに，トロイトリガーの多くが特定文字列に基づくことから，比較命令（例：`strcmp`）を重点的に探索できるよう，環境変数 `AFL_COMPCOV_LEVEL=2` などを設定し，比較命令に対する網羅性を高めた．

3.2.2 手順 2 システムコールの比較

手順 2 では，手順 1 で取得したテストデータを用いて検査対象と正常の両プログラムをそれぞれ実行し，`strace` を用いて実行時のシステムコールを収集する．これにより，PUT において探索された実行経路を正常なプログラムでも再現でき，トロイトリガーを含む実行経路を踏んだ場合，正常なプログラムとのシステムコールの差異からトロイ化を検出できる．また，テストデータごとに `strace` でシステムコールを取得し，両プログラム間でシステムコールの発行順序の差分を抽出する．なお，効率的にシステムコールを収集するため，新たに分岐を通過したテストデータのみを用いて手順 2 のファジングを行う．これは，すべてのテストデータを用いた場合，すでに実行した経路を繰り返し重複してファジングし，効率が悪化するためである．

3.3 提案手法の評価状況

我々の過去の研究 [4] では，提案手法の有用性を検証するため，バックドアデータセット ROSARUM の synthetic を用いた評価を行った．synthetic は，ライブラリや単体プログラムなど多種多様なプログラムに対して，第三者が意図的にバックドアを組み込んだ擬似的なソフトウェア群である．この評価により，提案手法が多様なプログラム構造やバックドアの種類に対し有効であることを確認した．

しかし，synthetic の評価はあくまで擬似的な環境下での有用性を示したものの，実際の攻撃事例として報告されているトロイ化に対する実践的な有用性は未検証である．

そこで，本稿では，ROSARUM の authentic を用いた評価を行った結果を示す．authentic は，過去に実際の攻撃で悪用された httpd や vsFTPd 等のネットワーク IoT 機器で動作するプログラムのトロイ化バイナリを含む．authentic に対する評価により，提案手法が現実の脅威に対しても有効であることを示し，この実践的な有用性を明らかにする．

3.4 提案手法の課題

本章で示した提案手法は，標準入力で動作する AFL++ を使用するため，authentic に含まれるソケット通信を前提とするネットワークプログラムにはそのまま適用できない．このため，authentic のプログラムを標準入力を通じて実行できるように提案手法の拡張が必要である．また，過去の評価 [4] における一部の synthetic では，手順 2 におけるシステムコール比較時に，乱数生成といった不確定な動作によってトロイ化に関係のないシステムコールが差分として出力され，多くの誤検出が発生した．このため，トロイ化に関係するシステムコールのみを絞り込む手法を検討し，その手法の有用性について再評価する必要がある．

提案手法における課題は以下の通りである．

課題 1： 提案手法の手順 1 において，ソケット通信を前提とするネットワークプログラムに対して標準入力を通じてファジングを実行できるようにする必要がある．

課題 2： 課題 1 と同様に提案手法の手順 2 において，テストデータを標準入力から与えつつ，システムコールを収集する必要がある．

課題 3： 提案手法の手順 2 におけるシステムコール比較時に，トロイ化に関係のあるシステムコールのみを比較する検出手法を検討する必要がある．

4. ネットワークプログラムへの対応手法

本稿では，3 章に示した提案手法 [4]（以降，従来手法とする）における課題 1〜3 およびその対処について述べる．

4.1 課題 1 への対処：AFL_PRELOAD の活用

課題 1 を解決するため，手順 1 のファジング処理において，`LD_PRELOAD` と同様の機能を持つ AFL++用の環境変数 `AFL_PRELOAD` を活用するよう従来手法を拡張した．これにより，事前に複数の共有ライブラリをリンクし，ソケット通信での処理を標準入力での処理に変換し，ファジングを可

表 1　比較対象のシステムコール

execve, fork, clone, ptrace, chmod, mprotect, setgid, setuid, accept, bind, connect, listen, socket

能にする．代表的なライブラリとして，preeny [18] が提供する共有ライブラリ desock.so があげられる．desock.so は，socket() のようなソケット関連の関数群を内部に持ち，これらはソケットを標準入力からのデータ読み込みに置き換えることができる．AFL_PRELOAD に desock.so を指定することで，ファジング実行時に desock.so を事前にリンクし，ファジング対象プログラムが例えば，socket() 関数を呼び出した際に，libc の本来の socket() ではなく，desock.so 内の socket() が実行されるようになる．

AFL_PRELOAD を活用し，テスト対象のソースコードを直接変更することなく，ソケット通信を行っているよう振る舞い，実際に AFL++が生成したテストデータを標準入力から受け取ることが可能となる．この拡張により，本来 AFL++でのファジングが困難なネットワークプログラムに対するファジングを可能にした．

4.2　課題 2 への対処：LD_PRELOAD の活用，および適切なオプションの設定

課題 2 を解決するため，手順 2 のシステムコールの比較において，strace が提供する-E オプションを活用するよう従来手法を拡張した．このオプションはトレース対象のプログラムに対して環境変数を指定するためのものであり，これを用いて LD_PRELOAD に手順 1 で構築した適切な共有ライブラリ群を設定する．この拡張により，strace の監視下でプログラムを実行する際にも，手順 1 と同様にソケット通信を標準入力へ変換することが可能となる．

4.3　課題 3 への対処：トロイ化に関係するシステムコールを絞り込んだ比較

従来手法をネットワークプログラムに適用した場合，トロイ化に関係のないシステムコールが差分として現れるため，多くの誤検出が発生する．この課題を解決するために，トロイ化に関係のあるシステムコールに絞った比較を行うよう従来手法を拡張した．文献 [19] では，不正機能において，不正機能が実行される際に，**表 1** に示すようなコマンド実行，権限変更，ネットワーク通信に関連するシステムコールが呼び出されることを報告している．我々の想定するトロイペイロードは，権限昇格や外部からのコマンド実行であり，表 1 に示すシステムコールに絞り込むことは有用であると考え，手順 2 のシステムコールの比較において，比較対象のシステムコールを絞り込み，誤検出を抑制する．なお，ネットワークプログラムに限らず，synthetic に含まれるコマンドプログラムにおいてもシステムコールの絞り込みを行い，誤検出を抑制できる．

表 2　提案手法の評価環境

CPU	Intel(R) Xeon(R) Gold 5120
メモリ	24.0 GB
OS	Ubuntu 24.04.2 LTS
ファジングツール	AFL++ 4.32c

5.　評価

5.1　評価方針

本評価では，ROSA [3] と同じデータセット ROSARUM を用いた比較評価を通じて，前章に記載した従来手法の課題に対処できるよう拡張した手法（以降，提案手法とする）が ROSA と同等以上の検出性能を有すること，および ROSA の課題を解決できることを示す．

評価項目として，以下の 3 つを設けた．

(1) トロイ化を検出できるか，および誤検出を起こすか
(2) 取得したシステムコールの差分情報からトロイ化の動作を特定できるか
(3) 単純なトロイトリガーによるトロイ化を検出できるか

5.2　評価環境

提案手法を評価する環境を**表 2** に示す．表 2 より，提案手法は ROSA と同様に，AFL++をファザーとして，ROSARUM をデータセットとして利用しているため，ROSA との比較評価が可能となる．ROSARUM は，バックドア検出ツールの評価を目的としたデータセットであり，多様なプログラムにおけるバックドア検出能力を測定できる．ROSARUM は 7 件の実在バックドア（authentic）と 10 件の合成バックドア（synthetic）を含んでおり，プログラム毎に以下の 3 種類のバージョンを提供している．正常なプログラム safe とトロイ化したプログラム backdoored が用意されているため，提案手法を適用できる．

(1) backdoored: バックドアが組み込まれている
(2) safe: バックドアが組み込まれていない
(3) ground-truth: バックドアを引き当てた時に，バックドアを検出した文字列を表示する

課題 1 および 2 への対処に関する評価では authentic，課題 3 への対処に関する評価では authentic および synthentic をそれぞれ用いた．authentic は，**表 3** に示すプログラム一覧の通り，トロイ化として IoT 機器のある特定のネットワークプログラムにバックドアが仕掛けられた状況を想定している．また，本評価では提案手法によりバックドアの動作を意図的に引き出すことを想定しているため，安全性を確保する観点から，Docker コンテナにより隔離された環境で提案手法を適用した．

5.3　評価方法

(1) **トロイ化の検出，および誤検出の評価：**以下に示す流れで，トロイ化の検出，および誤検出を評価する．

表 3　ROSARUM (authentic) に含まれるプログラムとバックドアの内容

プログラム名	種類	バックドアの内容
Belkin/httpd	ルータの HTTP サーバ	秘密 URL 値を含む HTTP リクエストによる web shell の接続
D-Link/thttpd	ルータの HTTP サーバ	秘密フィールド値を含む HTTP リクエストによる認証バイパス
Linksys/scfgmgr	ルータの TCP サーバ	特定のペイロードパケットによるメモリの読み書き
Tenda/goahead	ルータの HTTP サーバ	特定のペイロードを持つパケットがコマンド実行
PHP	HTTP サーバ	秘密フィールド値を含む HTTP リクエストによるコマンド実行
ProFTPD	FTP サーバ	秘密 FTP コマンドによる root シェルの接続
vsFTPd	FTP サーバ	":)" を含む FTP ユーザー名による root シェルの接続

(a) 新たに分岐を通過したテストデータを用いた ground-truth プログラムの実行を通じて，バックドア検出の文字列が表示されるかを確認し，バックドアのトロイトリガーを引き当てたテストデータを特定する．

(b) 提案手法でシステムコールが異なったテストデータが，特定したテストデータの中にあるかを確認する．

(c) テストデータがあった場合は検出成功と判断し，なかった場合は誤検出として判断する．

(2) **差分情報からのトロイ化の動作特定の評価：**提案手法と ROSA の適用時に，バックドアを引き当てたテストデータを用いたプログラム実行によって取得されたシステムコールの差分を分析し，この差分からトロイ化の動作を特定できるかを評価する．

(3) **単純なトロイトリガーによるトロイ化検出の評価：**2.3.2 項で示したように，authentic においてもトロイトリガーを簡単な条件に変更し，提案手法および ROSA を適用した結果を分析する．具体的には D-Link/thttpd のトロイトリガーを変更し，トロイ化を判定することができるか，および詳細な調査への影響を評価する．

5.4　authentic の評価結果

5.4.1　トロイ化の検出

表 4 より，7 件のプログラムに対して提案手法を適用した結果，すべてのプログラムにおいてシステムコールの差異を特定し，トロイ化を検出できた．一方，ROSA もトロイ化を検出できており，提案手法は ROSA と同程度の検出性能を有する．なお，Tenda/goahead のトロイ化は，トロイトリガーの前に，バックドアサーバの初期化処理を行うため，すべてのテストデータで差分が発生する．これはトロイ化にあたる処理であるため，バックドアを検出したという文字列が出力されなかった場合でも，検知可能とした．

5.4.2　トロイ化の誤検出

提案手法において，ProFTPD 以外のプログラムはすべて，誤検出率が 0.0%であった．また，ProFTPD における誤検出の原因は，`setuid()` や `setgid()` のシステムコールの発行順序が異なっていたという限定的なものであった．

一方，ROSA では Linksys/scfgmgr 以外のプログラムで，誤検出が発生し，原因は多様であった．vsFTPd の例では，バックドアを引き当てた可能性のあるテストデータにおいて，6 種類のシステムコールの差分を確認し，そのうちバックドアに関係する差分は 2 種類であり，バックドアに関係のない差分は 4 種類であった．

```
1  === backdoored にのみ存在するシステムコール===
2  2107750 socket(AF_INET, SOCK_STREAM, IPPROTO_IP) = 4
3  2107750 execve("/bin/sh", ["sh"], 0x5b5a65e70890 /* 14 vars
       ↪ */) = 0
4  2107750 mprotect(0x75396121d000, 2023424, PROT_NONE) = 0
5  // (中略)
6  2107750 mprotect(0x753961442000, 4096, PROT_READ) = 0
7
8  === safe にのみ存在するシステムコール===
```

図 2　提案手法で取得できるシステムコール差分情報

```
1  Found in the trace but not the cluster:
2  59
3  Found in the cluster but not the trace:
4  231
```

図 3　ROSA で取得できるシステムコール差分情報

以上より，提案手法は，ROSA よりもすべてのプログラムにおいて誤検出を抑えることができ，システムコールの絞り込みが 3.4 節の課題 3 への対処に有用であることが分かった．また，誤検出があった ProFTPD について，提案手法では誤検出の原因が限定的であり，ROSA のように多様な要因が関与する場合に比べ，調査工数が小さい．

5.4.3　差分情報からのトロイ化の動作特定

提案手法から得られるトロイ化の情報，および ROSA が詳細な調査を行うために提供する rosa-explain から得られるトロイ化の情報を比較し，どちらがトロイ化の動作を特定しやすいかを評価する．rosa-explain はバックドアの報告があったテストデータとそのテストデータが属する入力ファミリとのシステムコールの差分をシステムコール番号で表示するものである．今回は，vsFTPd を例に，バックドアを引き当てたテストデータにおける提案手法と rosa-explain の差分情報を，それぞれ**図 2** と**図 3** に示す．

図 2 より，提案手法はシステムコールの発行順序と引数を含めた差分を出力し，この時点で詳細な調査ができるようになっている．vsFTPd の例では，`backdoored` において，`socket()` システムコールによるソケット通信や `execve()`

表 4　提案手法と ROSA による検出可否，および誤検出率の評価結果

	提案手法		ROSA	
プログラム名	検出可否	誤検出率 *1	検出可否	誤検出率 *2
Belkin/httpd	✓	0.0% (0/3)	✓	96.9% (95/98)
D-Link/thttpd	✓	0.0% (0/68)	✓	93.5% (3,167/3,388)
Linksys/scfgmgr	✓	0.0% (0/248)	✓	0.0% (0/348)
Tenda/goahead	✓	0.0% (0/867)	✓	0.1% (1/784)
PHP	✓	0.0% (0/415)	✓	1.7% (329/18,588)
ProFTPD	✓	66.7% (4/6)	✓	100.0% (1,129/1,130)
vsFTPd	✓	0.0% (0/14)	✓	91.3% (137/150)

*1 提案手法における誤検出率：（バックドアを引かなかったテストデータの数／システムコールに差分が発生したテストデータの数）
*2 ROSA における誤検出率：（バックドアを引かなかったテストデータの数／バックドアの報告数）

システムコールによるコマンド実行を確認できる．

一方，図 3 における 1–2 行目は，バックドアの報告があったテストデータにのみ発行されたシステムコール番号を示し，3–4 行目は，入力ファミリにのみ発行されたシステムコール番号を示している．バックドアの報告があったテストデータのみに発行されたシステムコール番号 59 は，`execve()` システムコールであり，`strace` で vsFTPd のシステムコール番号 59 番の詳細な処理について調査することにより，バックドアと判定できるものの，提案手法で取得した `socket()` システムコールによる通信までは特定できない．このため，トロイ化の動作推定の精度は提案手法の方が優れていると推察できる．

5.4.4　単純なトロイトリガーによるトロイ化検出

過去の評価 [4] で，synthetic において評価を行い，単純なトロイトリガーの場合のトロイ化検出が可能であることを示した．本評価では，authentic に対しても有効であることを示すために，D-Link/thttpd に対して，トロイトリガーを単純にし，トロイ化を検出できるかを確認した．変更前後のトロイトリガーは**図 4** に示す通りであり，比較する文字列を短い単純なものに変更し，トロイペイロードには変更を加えなかった．ファジング時間は，5.3 節に示した評価方法と同様に 8 時間とした．

変更後の D-Link/thttpd に対して提案手法を適用した結果，発行されるシステムコールに差分があり，バックドアを検出でき，誤検出率は 0.0%(0/269) となった．一方，ROSA も同様にバックドアを検出できたものの，誤検出率は 77.6%(2,637/3,397) となった．単純なトロイトリガーへの変更により，ROSA はバックドア汚染を起こし，誤検出率が高い結果となった．一方，提案手法は，ROSA と比べて誤検出の増加を抑えている．さらに，提案手法で特定した差分は，通信関連のシステムコールとなり，D-Link/thttpd のバックドアの挙動と一致している．一方，バックドアを引き当てたテストデータに関して rosa-explain によって得られる情報は，トロイペイロードと関係のないものとなった．一例として，バックドアと関係のない `wait4()` システムコールに対応する番号 61 が出力された．これにより，

```
1 //変更前
2 iVar2=strcmp(hc->useragent,"xmlset_roodkcableoj28840ybtide");
3 if (iVar2 != 0) {
4 //変更後
5 iVar2=strcmp(hc->useragent,"test");
6 if (iVar2 != 0) {
```

図 4　ROSARUM D-Link/thttpd のトロイトリガーの変更

表 5　システムコール絞り込み適用後の提案手法の誤検出

提案手法（適用前）	提案手法（適用後）	ROSA
97.3% (1,278/1,313)	0.0% (0/35)	99.2% (5,250/5,295)

バックドアを引き当てたトロイ化の調査が困難になる上，見逃しが発生してしまう可能性が高いことがわかった．

5.5　synthetic の追加評価結果

synthetic に対しても，4.3 節に示したシステムコールの絞り込みの有用性を評価した．従来手法では，synthetic に含まれる 10 件のプログラムのうち，4 件で 70%を超える誤検出が生じていた [4]．一方，提案手法では，システムコールの絞り込みにより，すべてのプログラムで誤検出を 0.0%にできることを確認した．

次に，従来手法で特に誤検出率の高かった libxml2 の検証結果を**表 5** に示す．表 5 より，誤検出を削減でき，かつシステムコールに差分が発生したテストデータの数も大幅に絞り込むことができ，有用であることがわかる．

6.　議論

本章では，提案手法の正常な機器の入手可能性について議論する．ROSA は，検査対象機器に含まれる PUT のみで検知できるのに対して，提案手法は PUT に加え，正常な機器に含まれるプログラムを必要とする．さらに，正常なプログラムと PUT の差分は，トロイ化にあたるコードのみであるという制限がある．すなわち，正常なプログラムと PUT のバージョンが異なる場合や，PUT に独自パッチが適用済みの場合には誤検出が発生する可能性がある．

なお，正常なプログラムの入手方法は，2.1 節に示した

トロイ化のケースによって異なる．利用中に IoT 機器がトロイ化したケースは IoT 機器ベンダが Web サイトで公開しているファームウェアから抽出する方法や，OSS の場合は公式のリポジトリからの入手を想定している．サプライチェーン経由で IoT 機器がトロイ化したケースは，自社の開発環境からの入手を想定している．しかし，サプライチェーンの初期段階でプログラムに悪意のあるコードが混入し，それが全ての製品に組み込まれてしまった場合，比較対象となる正常なプログラムを入手できない．

7. 関連研究

IoT 機器における悪意ある振る舞いを検出する手法は，リストベース [20] や機械学習ベース [21], [22] の手法等，様々なアプローチが提案されてきた．リストベースの手法は，事前に IoT 機器の正当なプログラムやプロセスを定義し，それらの実行のみをリストで許可することにより，不正なプログラムの実行を検出する [20]．また，機械学習ベースの手法は，プログラムの正常動作時のシステムコール順序パターン [21] や CPU，メモリ，ファイルシステム等のカーネルイベント [22] を統計的に学習し，その学習結果に基づき異常な動作を検出する．これらの手法は，検査対象の動作と正常な動作を比較するホストベースな IoT 機器向けの手法という点で提案手法と共通するものの，本研究で対象とするトロイ化に対しては，トロイトリガーを引けず異常な動作を引き出せないことや，サンプル数が少ないため機械学習モデルの構築が困難といった問題がある．一方で提案手法は，機械学習を使用せずにファジングとシステムコールを組み合わせることにより，悪意ある振る舞いを引き出し，トロイ化を検出する．すなわち，IoT 機器配置前の事前確認として提案手法を，配置後の監視として上記関連研究を活用するよう補完する関係にあるともいえる．

8. おわりに

本稿では，バイナリファジングを利用した IoT 機器のトロイ化検出手法をネットワーク IoT 機器にも適用できるよう拡張し，その有用性を評価した．具体的には，AFL++ の環境変数および `LD_PRELOAD` を用いて必要な共有ライブラリを使用し，ネットワークプログラムにも適用可能にした．また，誤検出をさらに抑制するため，比較対象のシステムコールをトロイ化に関係するものに絞った．

評価では，ROSARUM の authentic を用いて ROSA と比較した．結果，提案手法と ROSA 両者とも authentic に含まれる 7 件すべてのトロイ化を検知できた．そのうち，提案手法は 6 件で誤検出率 0.0%を達成し，残る 1 件についても ROSA より誤検出を抑制できた．さらに提案手法は，ROSA よりもシステムコールの差分からトロイ化に関する多くの情報を得られ，トロイ化の動作特定性能も優れていることを示した．また，synthetic に対しても，システムコールの絞り込みの有用性を評価し，手法の拡張前後で最大 97.3%あった誤検出率を 0.0%に削減できた．

謝辞 本研究の一部は，岡山工学振興会特別研究の助成を受けたものです．

参考文献

[1] Antonakakis, M. et al.: Understanding the Mirai Botnet, *USENIX Security Symposium* (2017).
[2] Williams, L. et al.: Research Directions in Software Supply Chain Security, *ACM Trans. Softw. Eng. Methodol.*, Vol. 34, No. 5 (2025).
[3] Kokkonis, D. et al.: ROSA: Finding Backdoors with Fuzzing, *ICSE* (2025).
[4] 金城豪志，高田雄太，熊谷裕志，神薗雅紀，山内利宏：バイナリファジングを用いた正常なプログラムとの挙動比較による IoT 機器のトロイ化検出手法，情報処理学会研究報告， Vol. 2025-CSEC-110, No. 70, pp. 1–8 (2025).
[5] Noman, H. A. and Abu-Sharkh, O. M.: Code injection attacks in wireless-based Internet of Things (IoT): A comprehensive review and practical implementations, *Sensors*, Vol. 23, No. 13, p. 6067 (2023).
[6] Ladisa, P. et al.: Sok: Taxonomy of attacks on open-source software supply chains, *IEEE S&P* (2023).
[7] Ohm, M. et al.: Backstabber's knife collection: A review of open source software supply chain attacks, *DIMVA* (2020).
[8] Schuster, F. and Holz, T.: Towards reducing the attack surface of software backdoors, *ACM CCS* (2013).
[9] NIST: CVE-2024-3094 Detail , `https://nvd.nist.gov/vuln/detail/cve-2024-3094` (accessed 2025-05-27).
[10] Wen, C. et al.: MemLock: memory usage guided fuzzing, *ICSE* (2020).
[11] Kim, K. et al.: HFL: Hybrid Fuzzing on the Linux Kernel, *NDSS* (2020).
[12] Manès, V. J. et al.: The Art, Science, and Engineering of Fuzzing: A Survey, *IEEE Transactions on Software Engineering*, Vol. 47, No. 11, pp. 2312–2331 (2021).
[13] Michal Zalewski: American Fuzzy Lop, `https://lcamtuf.coredump.cx/afl/technical_details.txt` (accessed 2025-04-15).
[14] google: Honggfuzz, `http://honggfuzz.com/` (accessed 2025-04-18).
[15] Fioraldi, A. et al.: AFL++: Combining incremental steps of fuzzing research, *USENIX WOOT* (2020).
[16] Thomas, S. L. and Francillon, A.: Backdoors: Definition, deniability and detection, *RAID* (2018).
[17] Segura, S. et al.: A Survey on Metamorphic Testing, *IEEE Transactions on Software Engineering*, Vol. 42, No. 9, pp. 805–824 (2016).
[18] zardus : preeny, `https://github.com/zardus/preeny` (accessed 2025-08-07).
[19] Ghavamnia, S. et al.: Temporal System Call Specialization for Attack Surface Reduction, *29th USENIX Security Symposium*, pp. 1749–1766 (2020).
[20] Breitenbacher, D. et al.: HADES-IoT: A Practical Host-Based Anomaly Detection System for IoT Devices, Asia CCS (2019).
[21] Forrest, S. et al.: A sense of self for Unix processes, *IEEE S&P* (1996).
[22] Celdrán, A. H. et al.: Intelligent and behavioral-based detection of malware in IoT spectrum sensors, *Int. J. Inf. Secur.*, Vol. 22, No. 3, p. 541–561 (2022).