CL-001

組み込み機器におけるファジングの効率化手法の検討

Investigation of Methods for Enhancing the Efficiency of Fuzzing in Embedded Devices

小川 大賀 †　城間 政司 †　仲地 孝之 †　名嘉村 盛和 †
Taiga Ogawa　Tadashi Shiroma　Takayuki Nakachi　Morikazu Nakamura

1　はじめに

インフラ，ヘルスケア，スマートホームなど，様々な分野で IoT デバイスが普及しており，ファームウェアやアプリケーションを搭載したこれらのデバイスがサイバー攻撃の標的にされている．2016 年，Mirai ボットネットは数百万台の IoT デバイスを感染させ，大規模な DDoS 攻撃を行った [1]．現在も Mirai 亜種や新たな IoT マルウェアが出現しており，IoT デバイスのセキュリティ問題は重要視されている．

ソフトウェアの脆弱性検査の手法としてファジングというものがあり，IoT デバイスなどの組み込み機器に対してもその技術が応用されている．特にそのような組み込みシステム系のファジング技術は，アーキテクチャの多様化，クラッシュ検出の困難さ，限られたリソースなど，通常のデスクトップシステムに対するファジングとは異なる課題がある．例えば，多様なアーキテクチャの存在やハードウェア依存度の高さなどからエミュレーションによるファームウェアのファジングは困難な場合があったり，組み込まれたソフトウェアのソースコードが公開されていない場合ブラックボックス的なファジングを行う必要があり効率が悪い場合がある．

そこで，本研究では，検査対象のファームウェアの構成情報を利用してファジングを行う手法を提案する．具体的には，ファームウェアの静的解析を事前に行い，使用されている OSS やソフトウェアのリストを取得し，それらに対して AFL，libFuzzer などのツールを使ったカバレッジガイド型ファジングを行う．さらに，これらのファジング結果を利用し，ファームウェア全体へのファジングを行うことで効率化を図る．

† 琉球大学 University of the Ryukyus

2　関連研究

2.1　FirmAFL

FirmAFL[2] は，ファームウェアのファジングツールであり，エミュレーションの効率低下という課題に対処している．ユーザーモードとシステムモードの両方のコンテキストを持ち，必要時のみシステムコールを呼び出すことでファジング効率を向上させている．

2.2　EQUAFL

Yaowen Zheng らは，Linux ベースの IoT デバイス向けに設計されたアプリケーションに対する効率的な脆弱性検出手法として，EQUAFL[3] というフレームワークを提案している．EQUAFL は，上記 FirmAFL の手法をベースにしており，system-mode emulation と user-mode emulation の効率性を最大限活用することが主な目標になっている．また，EQUAFL では事前に収集した辞書ファイルを用いてファジングを行っている．

2.3　FIRM-COV

Juhwan Kim らは，ファームウェアのファジングにおいて，プロセスエミュレーションと独自の辞書生成手法を組み合わせることで，ファジングの効率を向上させる手法を提案している [4]．EQUAFL 同様に，system-mode emulation と user-mode emulation の両方を組み合わせることで，安定したファジングを実現している．また，ファジングの効率を向上させるために，ファームウェアの静的解析を行い readable strings(バイナリデータの中から文字として取得できるデータ) を抽出し，それを辞書とした mutation をファジングプロセスに組み込むことで，ファジングの効率を向上させている．

本研究では，FirmAFL および EQUAFL のフレームワークを用いて，コーパスをそれぞれ初期シードや辞書として活用する手法を実装し，FIRM-COV の辞書ベースファジングを拡張する新手法を提案する．具体的には，単に readable strings に依存するのではなく，組み

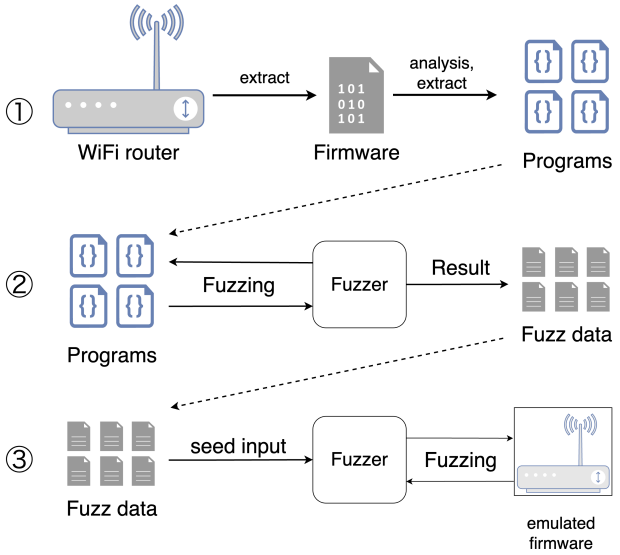

図 1 提案手法の概要図

込みライブラリやソフトウェアへのファジング結果を初期シードや辞書として活用することで，ファームウェア全体に対するカバレッジとファジング効率の向上を目指す．

3 提案手法

本研究では図 1 に示すように，WiFi ルーターや IoT カメラなどのファームウェアからソフトウェアの構成情報を取得し，取得した OSS やソフトウェアに対して AFL を使ったカバレッジガイド型ファジングを実施する．さらに，これらのファジング結果を利用し，ファームウェア全体へのファジングを行うことで効率化を図る．

以下で図 1 の各ステップについて説明する．

(1) ファームウェアのファイルを取得し，解析することで搭載されたソフトウェアのリストを生成，取得する．
(2) 取得したソフトウェアのリストを元に，それらに対して AFL++ などのカバレッジガイド型ファジングを行う．
(3) ソフトウェアに対するファジングの結果を活用して，ファームウェア全体へのファジングを行う．

このようにソフトウェアのファジング結果を再利用することで，直接ファームウェアをファジングするよりも多くの前提知識を持ってテストが可能となる．また、ファジング結果の再利用の方法としては、辞書を用いる方法とシードとして用いる方法の 2 つを検証する．

表 1 対象のファームウェア

device	fw ver.	executable	library
TV-IP110WN	1.2.2.68	video.cgi	libutl
DAP-2330	1.06	httpd	libcrypt

4 評価

4.1 評価方法

FirmAFL および EQUAFL を用いて，提案手法の有効性を評価する．FirmAFL では，コーパスをシードとして用いる方式について，EQUAFL では，コーパスを辞書として用いる方式について評価を行う．

提案手法の有効性を評価するため，以下の 2 つの指標を用いてファジングの結果を比較する．

1. 時間あたりのクラッシュ数：バグ（脆弱性）を引き起こす入力をどの程度効率的に発見できるかを示す．
2. 時間あたりの見つかったパス数：実行可能パスの探索をどの程度効率的に行えるかを示す．

提案手法によってファジングの効率が向上している場合，これらの値が増加することが期待される．

今回の実験では 24 時間にわたり 5 回ずつ実行と計測を行い，その結果を Andrea と Briand らのガイド [5] に基づき Mann-Whitney U 検定，Vargha and Delaney Â12 効果量を用いた結果の比較を行う．

評価に使用したファームウェアは表 1 に示す 2 つである．これらのファームウェアに対して事前に静的解析を行い，どのようなライブラリが含まれているかを特定し，それらのライブラリにに対して AFL++ を用いてファジングを行った．

4.2 コーパスをシードとしてを用いたファジング

FirmAFL を用いた Trendnet TV-IP110WN のファームウェアに対するファジングの結果を図 2 に示す．

図 2 は時間あたりのクラッシュ数と見つかったパス数の推移である．青の線は単一のシードを用いたファジング (凡例の fixed)，橙の線はコーパスを用いたファジングである (凡例の corpus)．実線はデータの平均，薄い色の領域は 95% 信頼区間を示している．パス数については，コーパスを用いた方式が固定のシードよりも多くなっていることがわかる．クラッシュ数については両者平均は 0 件であまり差は見られないが，コーパスを用いた方式の方が 1〜2 件分信頼区間が広い．

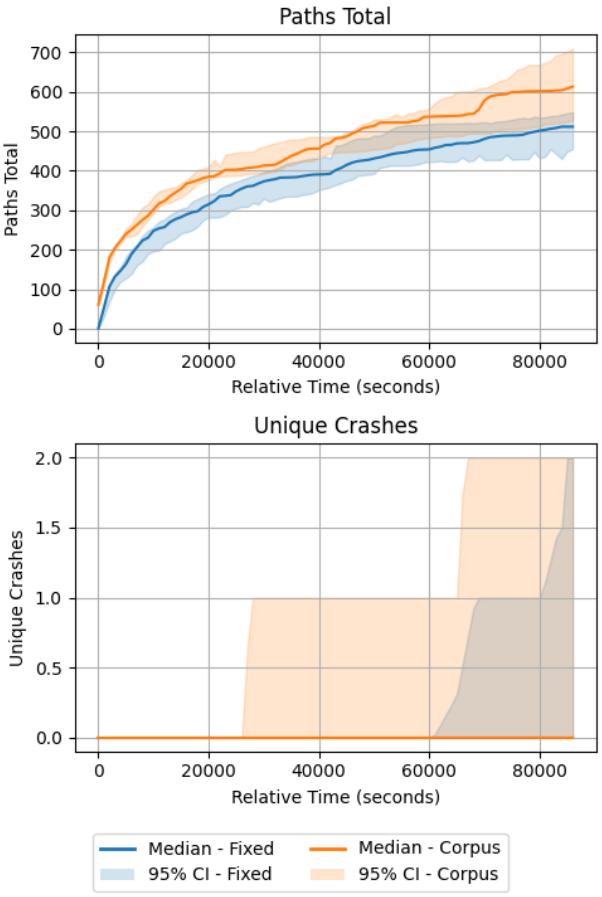

図 2 Trendnet TV-IP110WN のファジング結果

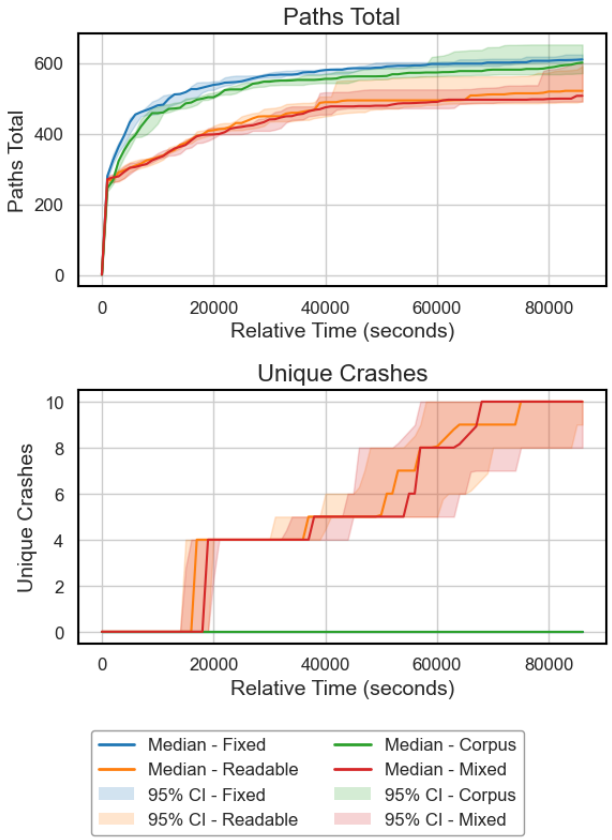

図 3 D-Link DAP-2330 のファジング結果

パス数について，コーパスを用いた方式は固定のシードを用いた方式と比べて A12 効果量の値が 0.86 と大きくなっており，また，U 検定の p 値も 0.010 であることから，有意な差があり，コーパスを用いたファジングのほうが効果的であると言える．

クラッシュ数については，コーパスを用いた方式と固定のシードを用いた方式の A12 効果量の値は 0.5 であり，U 検定の p 値も 1.000 であることから有意な差がないという結果が得られた．

結果として，シードとしてコーパスを用いたファジングは固定シードのファジングよりもパス数が多くなったが，クラッシュ数に関しては有意な差が見られなかった．

4.3 コーパスを辞書として用いたファジング

EQUAFL を使った D-Link DAP-2330 のファームウェアに対するファジングの結果を図 3 に示す．図 3 は時間あたりのクラッシュ数と見つかったパス数の推移である．実験パターンは以下の 4 つである．

1. 固定シード (青の線)
2. 固定シード ＋ readable strings (橙の線)
3. 固定シード ＋ コーパス (緑の線)
4. 固定シード ＋ readable strings ＋ コーパス (赤の線)

実線はデータの平均，薄い色の領域は 95% 信頼区間を示している．

グラフから，パス数に関しては辞書を用いていない方式と，コーパスを用いた方式が他の 2 つに比べて多くなっていることがわかる．クラッシュ数に関しては，コーパスのみを用いた方式で 0 件であり，readable string のみを用いた方式とコーパスを合わせて用いた方式のほうが明らかに多いことがわかる．

パス数に関して，コーパスを用いた方式は readable strings を用いた方式と比べて A12 効果量の値が 0.92 と大きくなっており，U 検定の p 値も 0.032 と小さくなっていることがわかる．しかし，辞書を使わない方式とコーパスを用いた方式は A12 効果量の値はむしろ 0.2 と小さくなっている．U 検定の結果からは，辞書を使わない方式とコーパスを用いた方式には p 値が 0.142 であることから有意な差が確認できないという結果が得られ

ている．

クラッシュ数に関しては，コーパスを用いた方式ではクラッシュが発生せず，readable strings および mixed のほうが多く見つけられる結果となった．

結果として，辞書としてコーパスを用いたファジングは従来の readable strings を用いたファジングよりも 100 程度パス数が多くなったが，クラッシュ数に関しては 0 件と，従来手法の 10 件に比べて少なくなった．

5 まとめ

本研究では，WiFi ルーターや IoT カメラなどの組み込み機器に含まれるファームウェアを対象に，内部構成を分析したうえで効率的にファジングを行う手法を提案した．具体的には，まずファームウェアに組み込まれている OSS やライブラリ単体を AFL でファジングし，その際に得られたコーパスをファームウェア全体（実行ファイル）に対するファジングへ再利用することで，コードカバレッジ向上やクラッシュ検出率の改善を狙う仕組みを構築した．

提案手法の狙いは，ライブラリ単体のファジングで抽出された入力データ群（コーパス）が，そのライブラリを内部で利用する本体の実行ファイルにも有用である可能性に着目した点にある．ライブラリの API や処理ロジックに対応した文字列や形式をあらかじめ取得しておくことで，ファームウェア全体をファジングするときに，ライブラリ経由で到達するコードパスへ効率よく入力を投げかけられると期待できるからである．

実験結果としては，提案手法が既存手法よりも一部，高いカバレッジを達成できたが，クラッシュ検出数については既存手法に劣る結果となった．

実験では，提案手法と既存の手法 (単一のシードや辞書として readable strings を用いた手法) を比較し，それぞれカバレッジの推移やクラッシュ検出数を評価した．その結果，提案手法ではシードとして用いた場合においては単一のシードを用いた場合よりも高いカバレッジを達成し，辞書として用いた場合は readable string を用いた場合と比べて高いカバレッジを達成できたが，辞書を用いない場合とあまり変わらないという結果が得られた．一方で，クラッシュ検出数という観点では，既存の手法のほうが提案手法よりも高いクラッシュ検出数を達成した．また，ライブラリから抽出したコーパスが実行ファイル内部で想定する入力形式と一部合わず，結果としてバグトリガになる入力パターンに到達しづらかったケースも考えられる．

今後は，ファームウェア全体のデータの流れをより正確に追跡し，ライブラリを通じて本体がどのように入力を処理しているのかを詳細に把握することで提案手法の有効性を詳しく検証することが必要である．

さらに，プログラムの実行をトレースしライブラリがどのように使われているかという調査は，デスクトップシステムなど解析がより容易な環境で実施した結果を組み込み機器へ応用できる可能性があるため，他の動的解析ツール・静的解析技術との連携も含め，今後の課題として継続的に検討していきたい．

参考文献

[1] Manos Antonakakis, Tim April, Michael Bailey, Matthew Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. Understanding the mirai botnet. In *USENIX Security Symposium*, 2017.

[2] Yaowen Zheng, Ali Davanian, Heng Yin, Chengyu Song, Hongsong Zhu, and Limin Sun. Firm-afl: High-throughput greybox fuzzing of iot firmware via augmented process emulation. In *USENIX Security Symposium*, 2019.

[3] Yaowen Zheng, Yuekang Li, Cen Zhang, Hongsong Zhu, Yang Liu, and Limin Sun. Efficient greybox fuzzing of applications in linux-based iot devices via enhanced user-mode emulation. *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2022.

[4] Juhwan Kim, Jihyeon Yu, Hyunwook Kim, Fayozbek Rustamov, and Joobeom Yun. Firm-cov: High-coverage greybox fuzzing for iot firmware via optimized process emulation. *IEEE Access*, 9:101627–101642, 2021.

[5] Andrea Arcuri and Lionel Claude Briand. A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering. *Software Testing*, 24, 2014.