Titanを活用した深層学習ファジング手法DeFuzzの改善

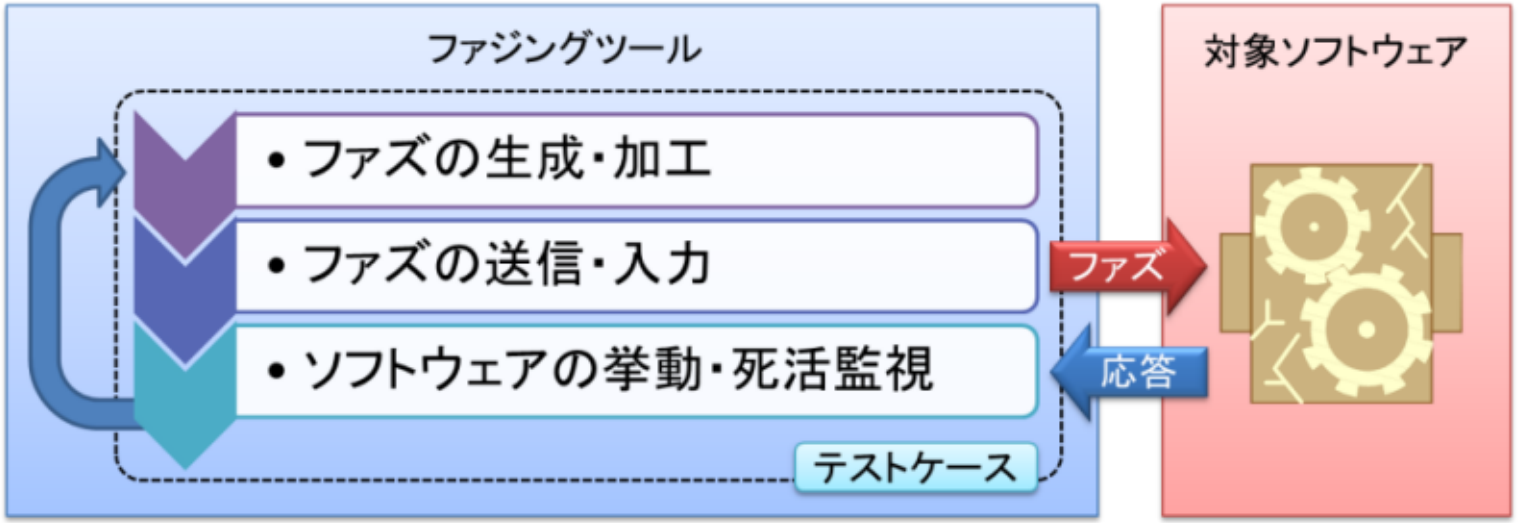
日本大学　文理学部
情報科学科　谷研究室
5421070　王潤政

目次

- 研究背景
- 先行研究
- 演習内容
- 実験

ファジング

・ファジングとは、コンピュータプログラムへの入力としてファズ(無効なデータ、予期しないデータ、ランダムなデータ)を用いた自動化又は半自動化されたソフトウェアテストの手法である。開発段階では予想していなかった入力データでテストできる。コンピュータプログラムにファズを与えることで、意図的に例外的な状況を発生させ、その例外的な状況での挙動を確認するという方法を用いる。



脆弱性

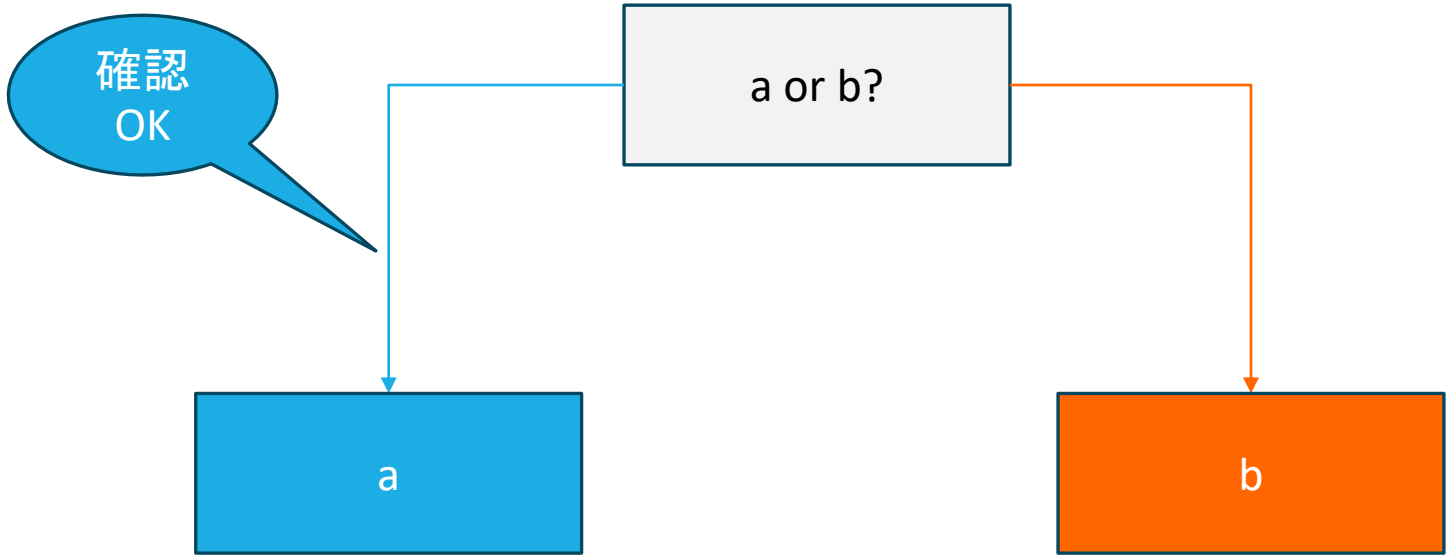
・脆弱性とは、コンピュータのOSやソフトウェアにおいて、プログラムの不具合や設計上のミスが原因となって発生するサイバーセキュリティ上の欠陥のことである。

・実際にはダブルフリーやメモリリークなどさまざまな脆弱性があるが、本演習では主に以下の2種類を対象とする。

- クラッシュ　メモリエラー等による異常終了
- ハング　処理の無限ループ等に起因するタイムアウト

コードカバレッジ

・コードカバレッジとはプログラムのソースコードがテストされた割合である。条件分岐が分岐aと分岐bの2つしかない単純なアプリケーションのコードで、条件付き分岐aをテストで検証した場合、分岐のコードカバレッジは50%と報告される。



目次

- 研究背景
- 先行研究
- 演習内容
- 実験

先行研究

- プログラムの脆弱性位置を検出するモデル
- 指向型ファジングツール：AFLGo
- 深層学習を用いたファジング手法：DeFuzz
- マルチターゲット特化する指向型ファジングツール:Titan

脆弱性位置の検出

・本演習が利用する脆弱性を検出モデルはまず収集したデータに基づいて深層学習モデルを訓練する。

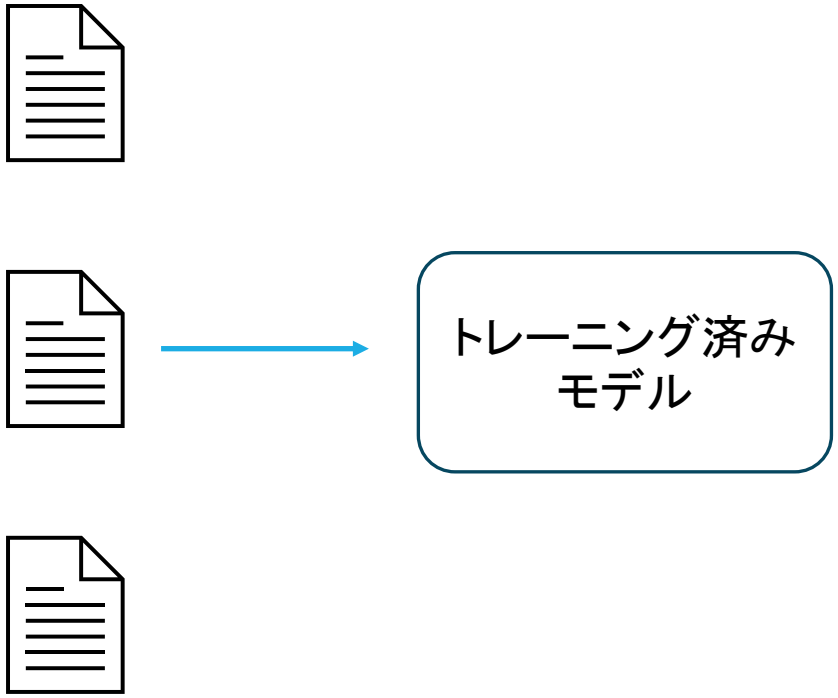
データサンプルはGitHub から収集する。NVDデータベースを参照して、関数に少なくとも1つの脆弱性が特定された場合、その関数は脆弱とラベル付けされ、それ以外の場合は脆弱ではないとラベル付けされる。

関数レベルでプログラムコードのAST表現を抽出する。ニューラルネットワークを使用して、特徴ベクトル表現に基づいて予測モデルを訓練する。

脆弱性位置の検出

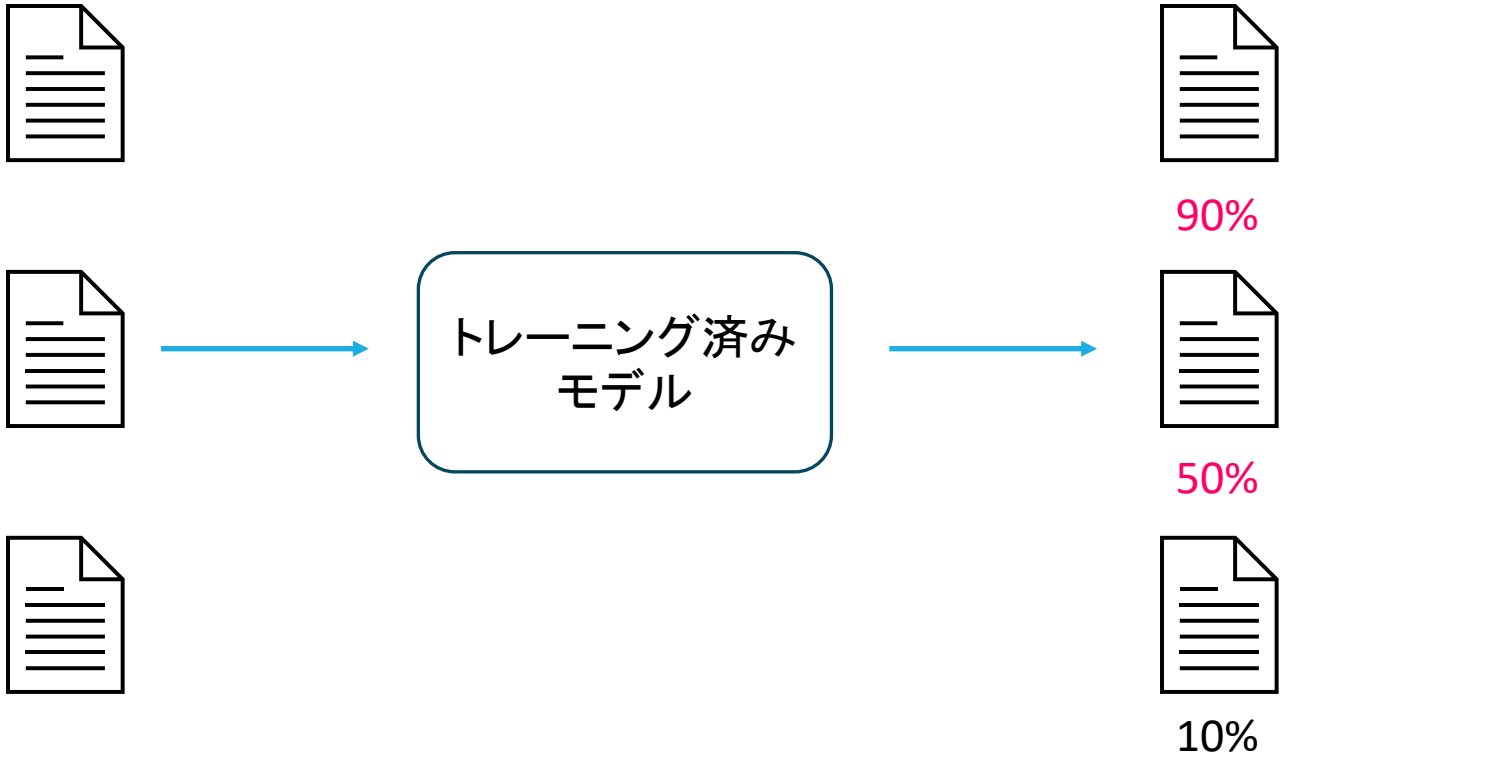
Input:プログラム関数

脆弱性位置の検出



Input:プログラム関数

脆弱性位置の検出

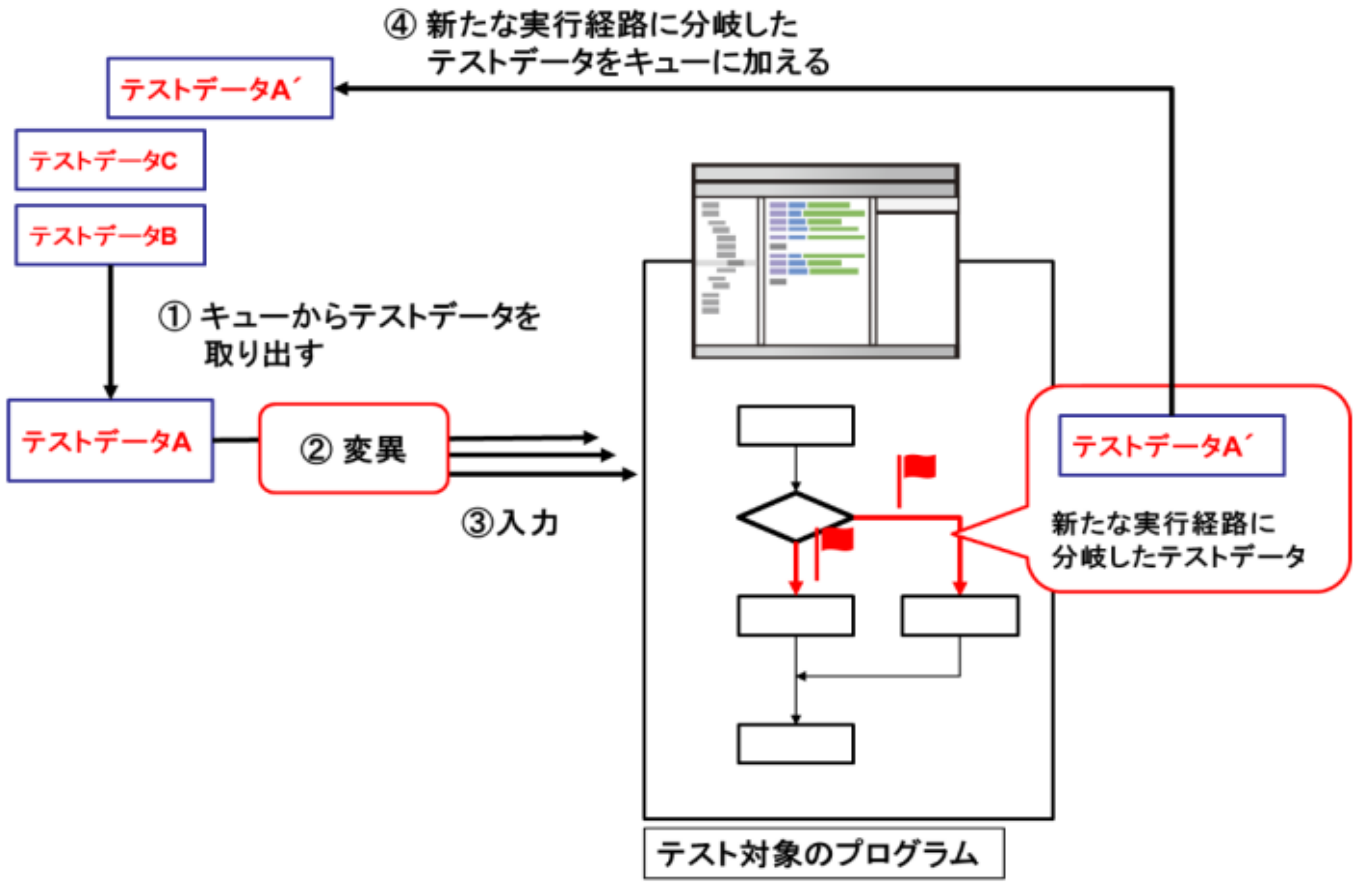


Input:プログラム関数

Output:関数が脆弱性の確率

AFL

・AFLはプログラムの実行経路を計測し、新たな分岐が見つかったらそのテストデータを保存して再利用することで、効率的にコードのあらゆる部分をテストできるファジングツールである。これにより、単純なファジングよりも高いコードカバレッジを達成することが可能。



AFLGo

・AFLGoは特定のターゲット位置（脆弱性位置）に重点にテストケースを生成する指向型ファジングツールである。以下の三つの段階で動作する：

・「計算」段階
　グラフに基づいて各ブロック(BB)からターゲット位置までの距離を計算する。

・「探索」段階
　局所解を避けるために、AFLGoのファジングが先ずは全てのテストケースを平等に変異チャンスを与える。

・「利用」段階
　距離を利用し、各テストケースのエネルギーを決定する。ターゲット位置との距離が小さいほど、エネルギーが大きく、多くの変異チャンスが割り当てられる。

AFLGoの変異戦略

bitflip	排他的論理和を用いた変換を行う
arithmetic	数字加算/減算による変換を行う
interest	テストデータに固定値を挿入する変換を行う
havoc	16種類のランダムな変換方式を繰り返す
splice	他のseedファイルと合弁して、havocをもう一回行う

AFLGoによるエネルギーがこの段階の時間をコントロールする

DeFuzz

深層学習モデル

収集したデータに基づいて深層学習モデルを訓練し、プログラム内の脆弱性が存在する可能性の高い位置を特定する。

＋

指向型ファジング

指向型ファジングツールAFLGoを使用して、脆弱性が特定された位置に近いテストケースをより多く変異させてファジングを行う。

＝

DeFuzz

一般的なファジングツールはできるだけ多くのカバレッジを発見することを目指している。仮定として、カバレッジが広いほど、脆弱性が露呈する可能性が高くなるとされている。しかし、これが常に正しいとは限らない。そのため、AFLGoのような指向型ファジングは、潜在的な脆弱性の場所に近いパスを探索することを優先して多くのバグ発見することが期待している。

従来手法の問題点

・非効率なテストケース選択

AFLGoはすべてのターゲットへの距離に基づいてテストケースを選択すると、他のターゲットに到達するのが困難になる可能性がある。

・変異の方向が不明

入力データとターゲットの正確な関係を決定することは非常に困難のため、ターゲットと無関係な分岐をカバーすることで無意味な努力が生じ、指向型ファジングの効率を低下させる。

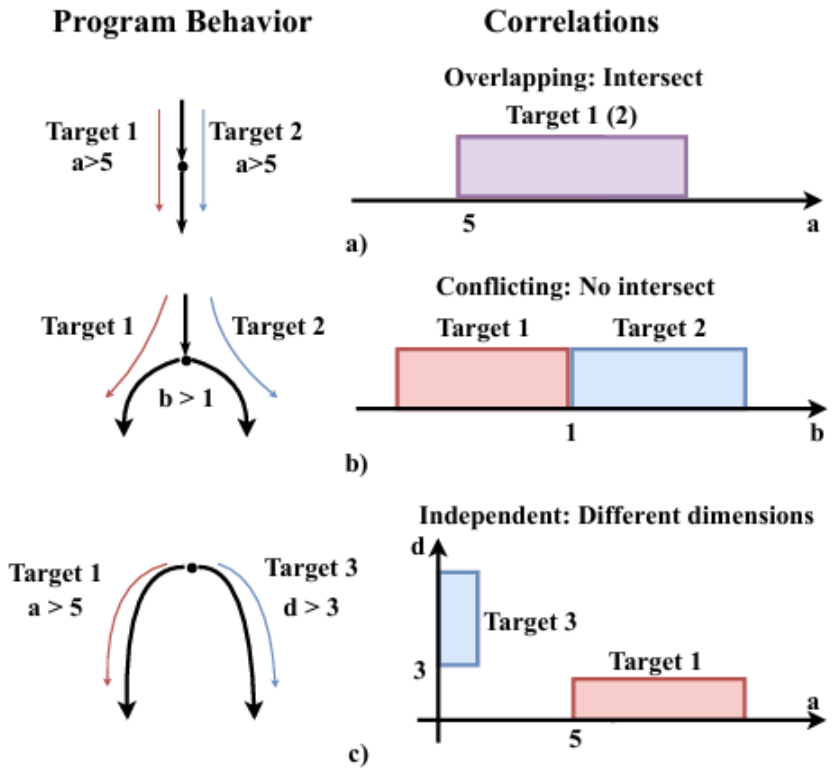
Titan

・Titanはマルチターゲットに向けてターゲット間の関係も静的解析の際に考える。

・重複条件：条件 a > 5 はターゲット1とターゲット2の両方に対する重複条件であるので、この分岐をカバーする可能性が高いテストケースを優先する。

・対立条件：ターゲット1とターゲット2に到達するためには、互いに対立条件がある。b < 1 を満たすシードはターゲット1をカバーする可能性が高く、b > 1 を満たすシードはターゲット2をカバーすることに重点する。

・独立条件：d > 3 はターゲット3の到達可能性にのみ影響を与え、a > 5 はターゲット1とターゲット2の到達に影響を与える。d と a を同時に変異することで、単一の変異で3つのターゲットすべてにアプローチすることが可能になる。



目次

- 研究背景
- 先行研究
- 演習内容
- 実験

演習目的

・ファジングテストは、実際の使用ではすべてのコードカバレッジに到達するのが難しい。現状のファジング手法は、状態空間を迅速かつ広範に探索できないため、バグの検出には非常に時間がかかる。そこで、ディープラーニングの利点を活かし、バグの識別のためにディープラーニング指向のホワイトボックスファジングを開発する。

・DeFuzzが深層学習モデルを用いて特定した脆弱性位置は通常複数存在するため、AFLGoは複数のターゲットに対して効果が十分に発揮されない問題がある。また、全てのターゲットに対して個別にファジングを行うには非常に多くの時間が必要であり、現実的な手法ではないという課題がある。

Titan

・HuangらはAFLGoに基づき、複数のターゲットに特化したファジングツールであるTitanを提案し、既存の二つの指向型ファジングツールAFLGoとBeaconよりも優れた性能を持つことを実証した。

演習内容

・本演習では、論文に基づいてDeFuzzというファジング手法を再現し、その上でDeFuzzにおいて深層学習モデルが検出した脆弱性位置に対してファジングを行うAFLGoの機能をTitanに代替することで、DeFuzzの改善を目指す。

・ Titanのマルチターゲット対応機能を活用し、深層学習モデルが特定した複数の脆弱性位置に対して効率的にテストケースを生成することで、より迅速な脆弱性発見を目指す。これにより、従来の手法に比べて検出率の向上とテスト時間の効率化が期待される。

目次

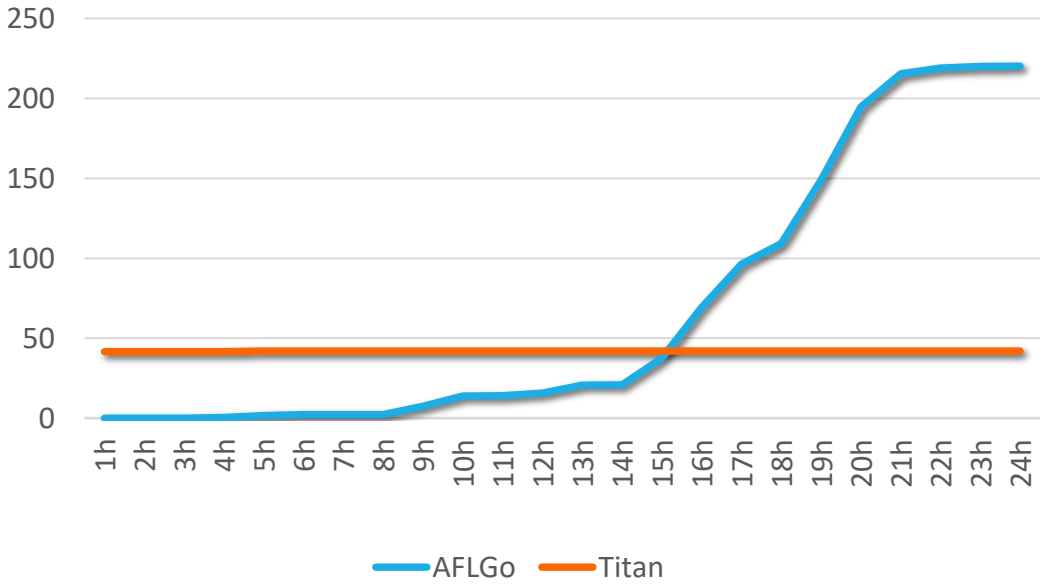
- 研究背景
- 先行研究
- 演習内容
- 実験

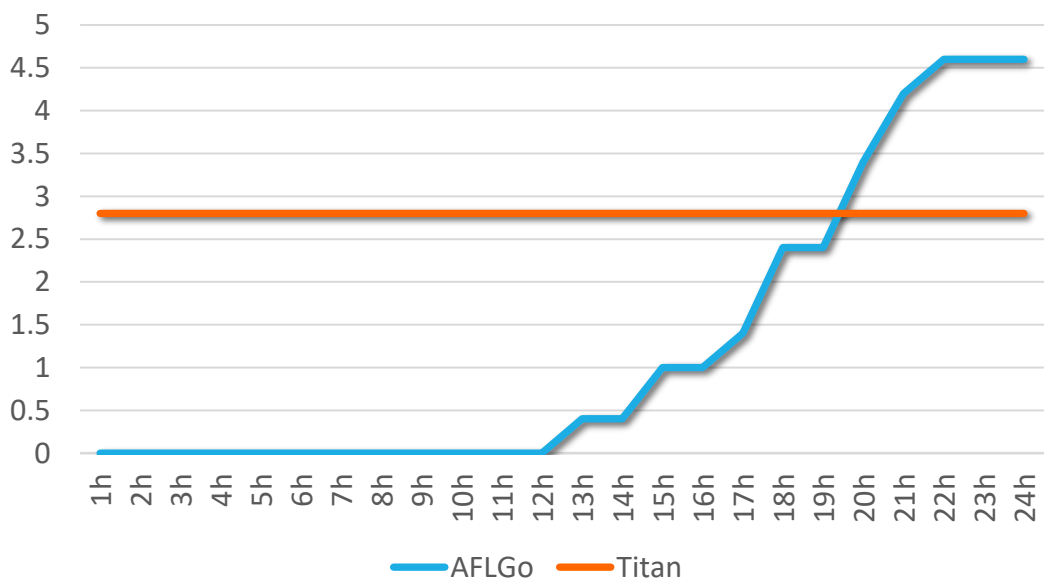
評価実験

・本演習では提案した手法を適用したDeFuzzと未適用のDeFuzzを実装し、比較実験を行う。AFLGoの「探索」段階から「利用」段階に移行するタイミングは20時間、ファジングの打ち切り時間は24時間とする。各手法はlibpngとlibtiff2つのオープンソースプログラムに対してファジングを行い、それぞれの実験は5回実施する。

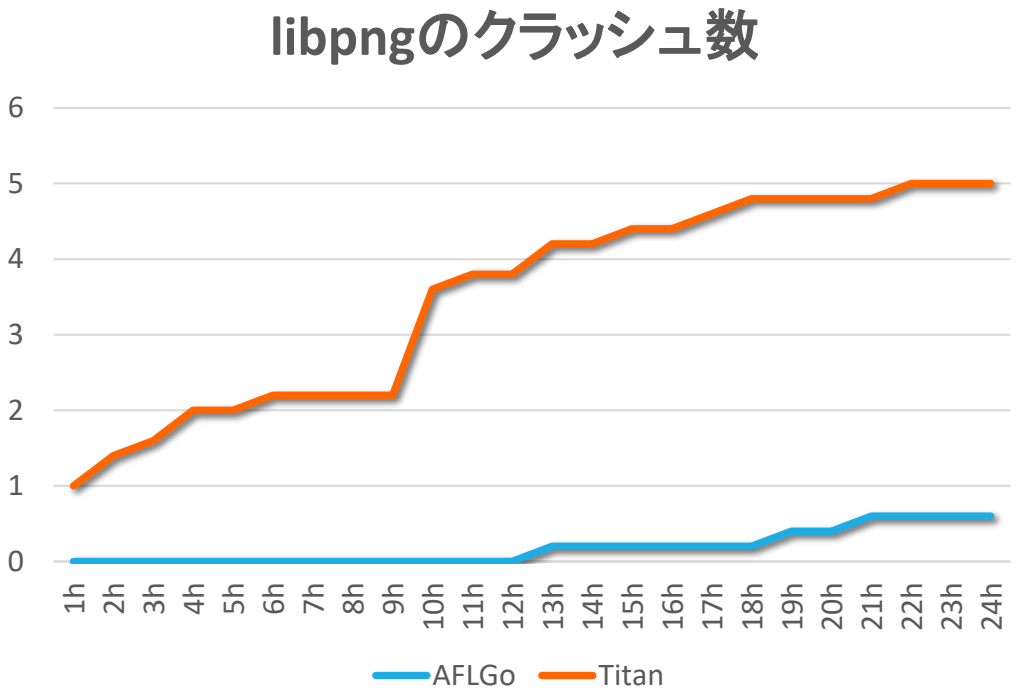
・評価指標としては、一時間ごとにファジングによって検出されたクラッシュとハング数の平均値2つのデータを用いて、提案手法の有効性を検証する。

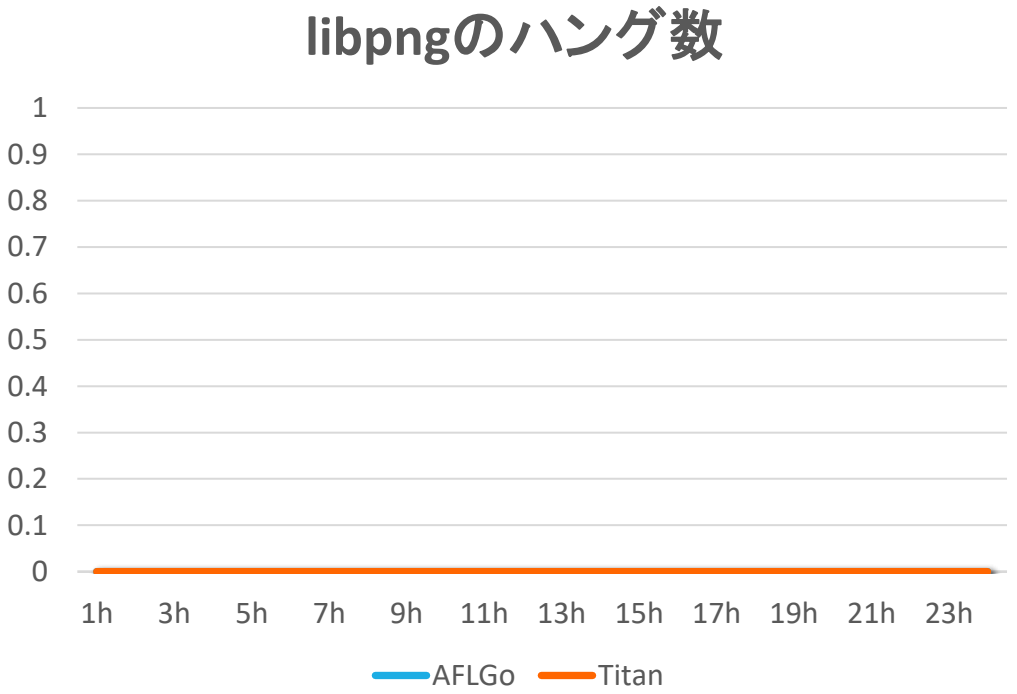
実験結果


libtiffのクラッシュ数
250
200
150
100
50
0
1h
2h
3h
4h
5h
6h
7h
8h
9h
10h
11h
12h
13h
14h
15h
16h
17h
18h
19h
20h
21h
22h
23h
24h
AFLGo
Titan


libtiffのハング数
5
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
1h
2h
3h
4h
5h
6h
7h
8h
9h
10h
11h
12h
13h
14h
15h
16h
17h
18h
19h
20h
21h
22h
23h
24h
AFLGo
Titan

実験結果


libpngのクラッシュ数
6
5
4
3
2
1
0
1h 2h 3h 4h 5h 6h 7h 8h 9h 10h 11h 12h 13h 14h 15h 16h 17h 18h 19h 20h 21h 22h 23h 24h
AFLGo
Titan

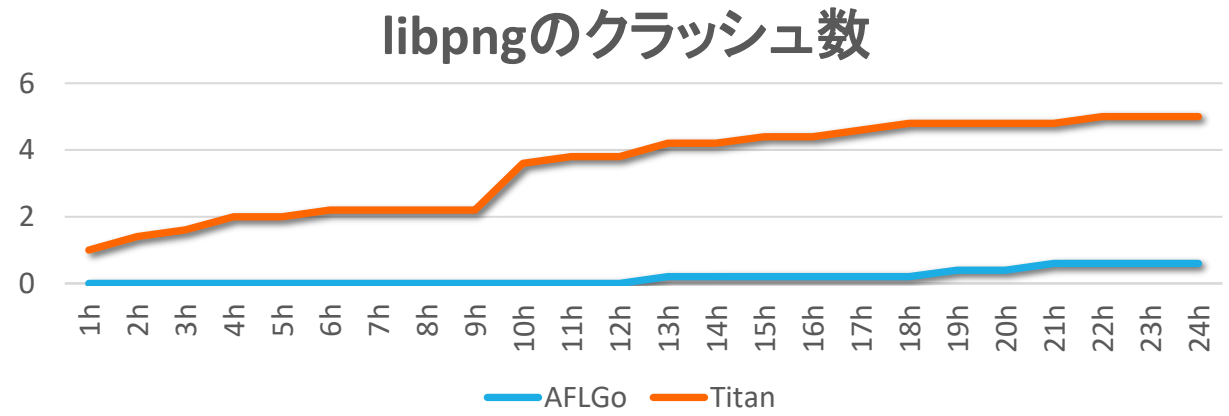

libpngのハング数
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
1h 3h 5h 7h 9h 11h 13h 15h 17h 19h 21h 23h
AFLGo
Titan

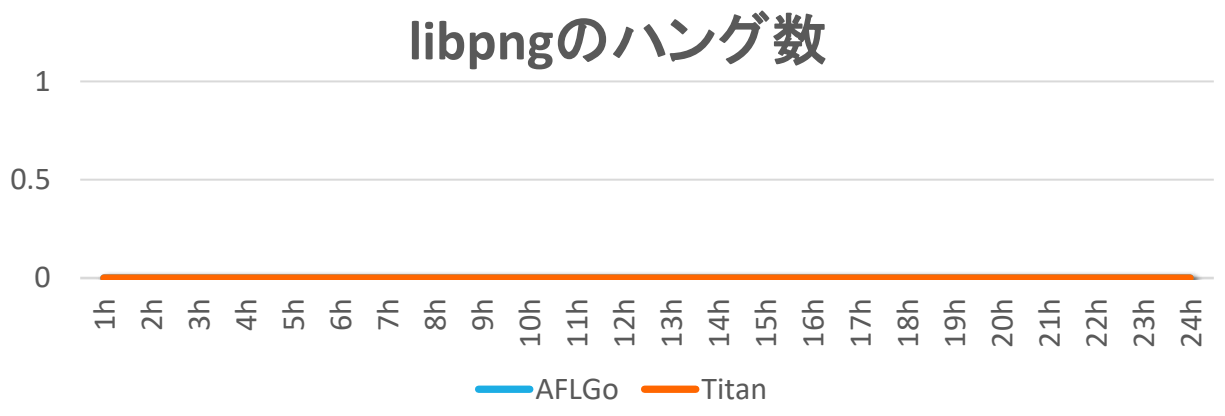
実験結果

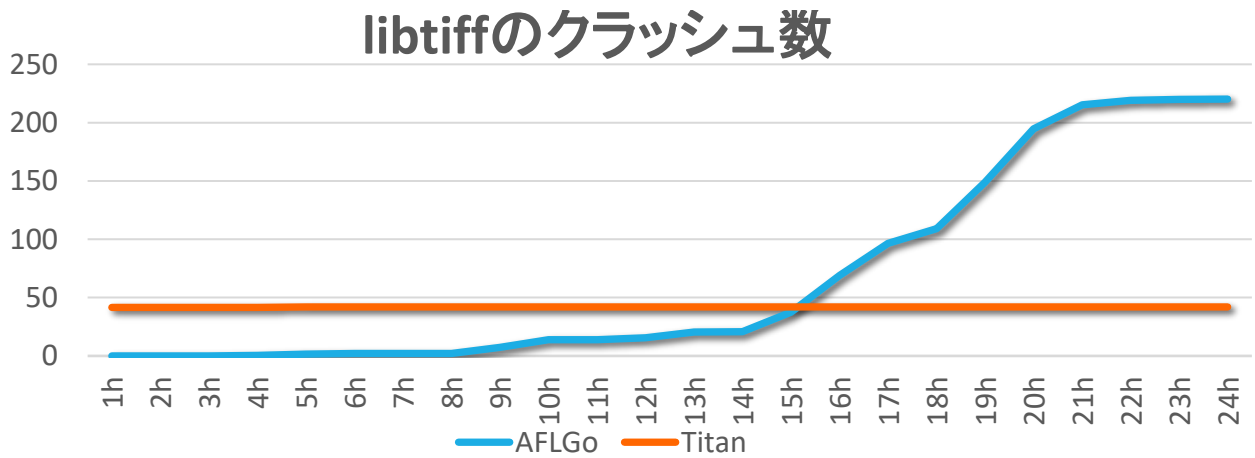
	libtiff						libpng					
	クラッシュ			ハング			クラッシュ			ハング		
	1h	3h	24h	1h	3h	24h	1h	3h	24h	1h	3h	24h
AFLGo	0.0	0.0	220.2	0.0	0.0	4.6	0.0	0.0	0.6	0.0	0.0	0.0
Titan	41.6	41.6	42.0	2.8	2.8	2.8	1.0	1.6	5.0	0.0	0.0	0.0

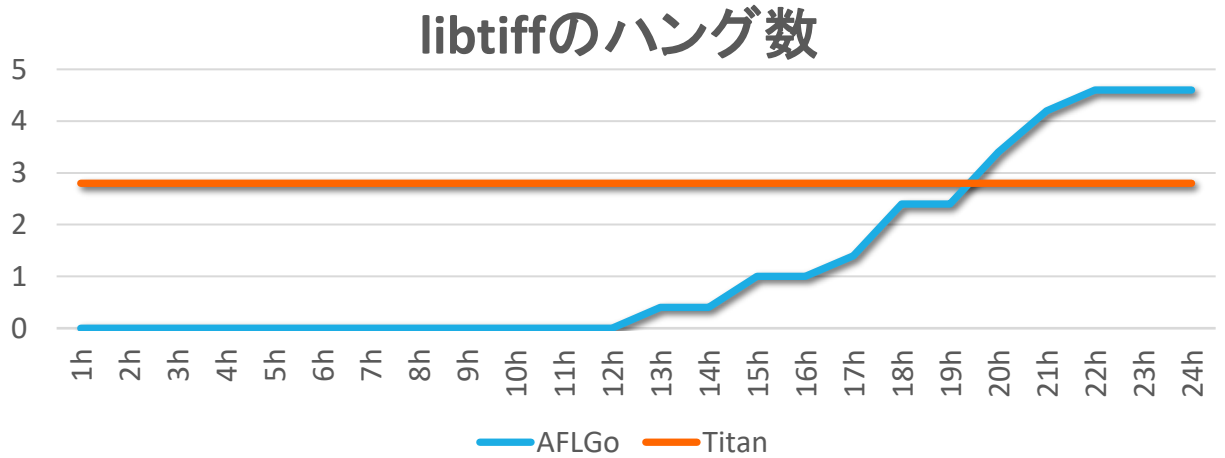
・小数点第2位を四捨五入

考察









・実験結果によると、適用したDeFuzzは未適用のDeFuzzと比較して1回の実行で複数のターゲットをより効率的にファジングできるようになった。しかし、一部のバグは単なるパス条件を超える複雑なセマンティクスを伴う可能性があり、そのようなバグを発見するにはコードカバレッジを高める方が有効な場合もある。効率とコードカバレッジのトレードオフはまだファジング分野の未解決の課題である。

今後の課題

・本演習では、トレードオフを直接解消するのではなく、機械学習とファジングを組み合わせる可能性に注目している。今後の研究では機械学習を活用してプログラム動作に対する動的解析を組み込み、より高度な検出手法やテスト効率の向上を目指す予定である。