Advanced Program Analysis Seminar

Koushik Chowdhury
2572865

Abstract

Program analysis is the procedure of analysing the performance of a program. It checks the correctness, robustness of a program. This report describes the high-level idea, summarization and connection of 12 states of art program analysis research papers. These research papers are grouped into five categories such as Dynamic Analysis, Abstract Interpretation, Correctness, Application and Beyond. This report summarizes the idea and approaches of these 12 papers and tries to find the limitations and connections between the papers in each group. The approaches include strength, applicability, limitations, etc. Some of the papers are from dynamic analysis and some are from static analysis. Each paper implements the different application for program analysis and each paper tries to improve the performance of the program

1 Introduction

Programmer writes the program. The program can be anything. It can be java program, C program or C++ program. To checks the program's correctness, robustness, reliability, Program analysis is important. Program analysis concerns the optimization and correctness of the program. With optimization, it is possible to avoid useless computation of the program and with correctness, it is possible to verify the assurance of the program. Program analysis helps to improve the performance of the program. There are mainly two types of program analysis. Static program analysis and Dynamic Program analysis. Static analysis is performed without being executing the program where dynamic analysis execute the program [1]. Static program analysis is less precise than dynamic program analysis. In this Report, 12 program analysis paper has been discussed. Dynamic analysis paper such as Driller, DGF, Automatic testing, DroidRacer and conditional model checking have been included and other papers such as AI^2, OptOctagon, Static probabilistic, Varasco, RacerD, FlowDroid and Contract-Based Verification are from static analysis.

2 Dynamic Analysis

Fuzzing technique involves providing random data as input to a program for monitoring the exception of the program. There are 3 main fuzzing techniques such as Black-box, white-box and grey-box fuzzing. The report concerns the grey-box fuzzing. Some researcher from the National University of Singapore has presented directed GreyBox Fuzzing (DGF)[2]. They combine grey box fuzzing and simulated annealing. Simulated Annealing solves the unconstrained and bound-constrained problem from optimization[3]. They implement DGF as AFLGo which is available in the public git [4]. AFLGo combines as patch testing tool into OSS-Fuzz. They show the DGF techniques to three application such as patch testing, continues fuzzing and crash reproduction. But before that, the researcher implemented the DGF technique into undirected greybox boxing which is known as AFLGo. Then the researcher applied AFLGo to patch testing and crash reproduction, and combines it with OSS-Fuzz[5]. There is four component in AFLGo such as graph extractor, distance calculator, instrumentor and AFLGo fuzzer. In the beginning, Graph Extractor takes the program source code and produce all graph and control flow. Then the distance calculator occupies

the call graph and control flow graph to measure the distance of each basic block and then the instrumentor specifies the target distance for basic block and make a binary file which has 64kb shared memory. In the end, the fuzzer tries to find the error input by taking the seed inputs. The researchers have shown this technique to patch testing and then compare it with the KATCH. KATCH integrates the symbolic execute with various heuristic to test the patches [6]. The researchers compare the result in the respect of 'Binutils' and 'Diffutils'. The AFLGo has found 223 basic blocks where KATCH has covered 198 basic blocks. Also, there are 810 uncovered changed basic blocks and 1018 changed basic blocks.

	#Changed Basic Blocks	#Uncovered Changed BBs	KATCH	AFLGo
Binutils	852	702	135	159
Diffutils	166	108	63	64
Sum	1018	810	198	223

Figure 1 Patch coverage results

Therefore, the observation says, in the terms of patch coverage, AFLGo outperforms the KATCH. The researchers also compare the vulnerability detection between AFLGo and KATCH where they have found more unreported bug in AFLGo than KATCH. Here, again, the AFLGo has given the best statistics. The researchers have studied the continuous fuzzing application and observed that for OSS-Fuzz, the AFLGo is successful than any other patch testing tools. To the Directedness of AFLGo, the researchers have compared the AFLGo with AFL and have found that AFLGo is directed and effective and also the best option as a crash reproduction tool. The researchers have made an experiment the AFLGo with BugRedux and they have found that BugRedux is less effective than AFLGo. There is some limitation of this research such as AFLGo does not cover many basic blocks due to the limitation of the prototype of AFLGo. Some researchers from UC Santa Barbara have presented a hybrid vulnerability excavation tool named Driller to find the deeper bug. Driller borrows fuzzing and selective concolic execution to find the deeper bug [7]. UC Santa Barbara researchers experiment Driller on 126 single binary application to the effectiveness. Driller consists of several components such as Input test cases, fuzzing, concolic execution, repeat, etc. Concolic execution is a verification technique and performs symbolic execution [8]. The researchers have run their evaluation in AMD64 processors. They have found 68 crashes when they applied fuzzing method. There have also found the same numbers of crashes when they combine Fuzzing and Driller. On the other side, Driller has found 77 crashes which are bigger than any of the method described in the paper.

Method	Crashes Found
Fuzzing	68
Fuzzing ∩ Driller	68
Fuzzing ∩ Symbolic	13
Symbolic	16
Symbolic ∩ Driller	16
Driller	77

Figure 2 Experimental Results[6]

They have observed that Driller can identify bug than any other specific method or combined method. The researchers have shown the numbers of concolic execution was invoked for these single binary application. They have found that number of binaries decreases when the number concolic execution invocations increase. Like the DGF paper, this paper has also limitations. The Driller cannot work effectively if the user gives input as generic in one component and specific in another component. In this case, fuzzer robes Driller of its advantage. There is also some limitation in the source of state-space representation. Driller borrows state-space representation from the AFL. Beyond all the limitation, according to the researcher, Driller is the best vulnerability excavation tool that effectively finds bugs in a binary.

3 Dynamic Analysis:Discussion

The DGF paper mainly focuses on the cover program path and shows some comparison between AFLGo and KATCH. On the other hand, Driller is a tool that finds bugs. As both techniques are based on system execution, therefore both are dynamic. The main advantage of dynamic analysis is that it finds the vulnerabilities as both paper talks about these issues. The main limitation of dynamic analysis is that it does not give guarantee the full test coverage. DGF paper concerns the directness and to do that the researchers implemented the AFLGo. AFLGo is mainly the extension version AFL. They have shown in the paper that AFLGo can outperform existing symbolic-execution based whitebox fuzzer. Driller paper mainly focuses on the 'Driller' tool that mainly used to find deeper errors as we all know the corruption in the memory vulnerability is common in the software. The most common keyword from these two paper is fuzzing where one paper tries to implement the Greybox fuzzing that can be directed and another paper concerns the augmentic fuzzing. The Driller paper is more practical than the DGF paper as Driller applied on 123 application to show its efficiency through both papers is competitive in the fuzzing field.

4 Abstract Interpretation

Static analysis is performed without executing the program. Static analysis involves abstraction [**?**]. Abstract interpretation can be inspected as a partial execution of a program. Abstract interpretation gain information about its semantic such as control-flow and data-flow [10].Abstraction helps to find interesting properties of computer programs. Some researchers from ETH Zurich present AI2 [11]. It's a scalable analyzer for neural networks. The neural network is one of the most used techniques in statistical learning. Many researchers have shown the way and the purposes of the neural network for different fields for their research. AI2 proves the safety properties such as robustness of neural networks such as CNN [11]. CNN or convolutional neural network is popular in image analysis. The researchers illustrate AI^2 together with 20 neural networks. They have presented that AI^2 can prove the robustness of neural network defenses. They have shown that AI2 is faster analyzer and it can handle the CNN. The researchers have used MNIST and CIFAR-10 datasets to make the evaluation where both consists of images. MNIST has grayscale images with 28x28 pixels and CIFAR has colored photographs with 32X32 pixels. The researchers have applied CNN and FNN on MNIST and CIFAR. They have found that each network had a test set accuracy of at least 90%. They have selected 10 images from MNIST and CIFAR and then specify a robustness property for every image. They have applied AI2 on these images to check whether CNN and FNN satisfy the robustness property or not and finally the compare their result with abstract

domains. There are few abstract domains are available: Box, Zonotope and Polyhedra where Polyhedra's abstract transformers are less faster than Box's abstract transformer. The results are following.

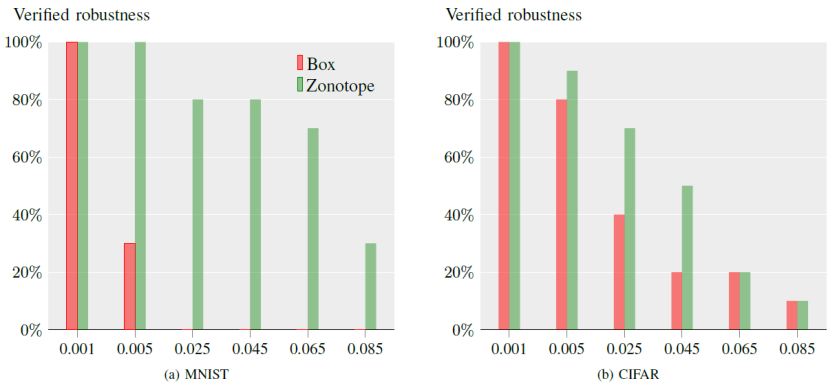

Figure 3 Verified properties by AI2 on the MNIST and CIFAR[10]

For both MNIST and CIFAR, AI2 checked robustness for the smallest bound 0.001 and for the larger bound 0.085. This proves that AI2 can verify properties for CNN and it is the first system that able to verify convolution and connected networks. Some other researchers from ETH Zurich present a paper on numerical program analysis [12]. They have shown an approach for optimizing the octagon domains. The main purpose of the paper is to improve the time performance of numerical abstract domain such as octagon domain. Octagon abstract domain is a numerical domain for static analysis by abstract domain [13]. As a part of the APRON C library, they have implemented new octagon operators. Octagon operator increases up the analysis time of a static program. The researchers have introduced a new data structure to the octagon as well as they have developed a new algorithm to compute octagon closure. The closure operator reduces the operation time of a program. They have experimented their approach in the APRON C library. They have replaced the operator of APRON C with the operators of their new algorithm. They called this version as OptOctagon.They have used a high configured CPU (Intel Core i7-4771 Haswell) for the experiment. They have used four static analyzers and every analyzer they have reported benchmarks that had at least 20 variable. They have tested more than 40 benchmarks where some of them have more than 10k LOC. Later, they have compared the result with Floyd-Warshell and OptOctagon

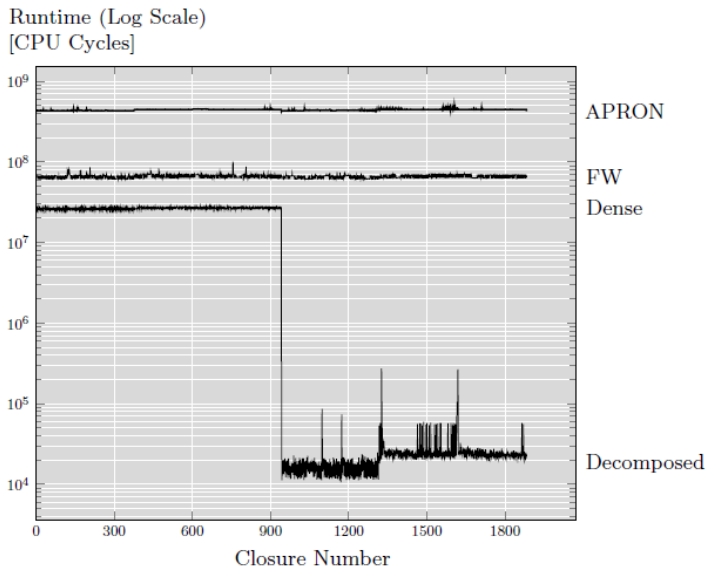

Figure 4 APRON vs Floyd-Warshell vs OptOctagon[11]

It has been found that Floyd-Warshell is 7-8 times faster that APRON, and OptOctagon is 3 times faster than Floyd-Warshell. It can be said that OptOctagon is the faster static numeric program analysis. Few researchers from the University of Colorado have presented static analysis for Probabilistic Programs. To do that, they have observed the infinite static program that manipulate uncertain quantities defined by probability distributions [14]. The probabilistic program describes probabilistic models. The researchers have concerned the semantic probabilistic programs. Probabilistic program represents the probabilistic model and that probabilistic models are flexible. They evaluate their static program over the set of benchmarks [15]. They have found the following experimental results.

ID	#var	Description
EGFR-EPI	11	log egfr calc. using the ckd-epi formula.
ARTRIAL	15	Framingham artrial fibrillation risk calculation.
CORONARY	16	Framingham coronary risk calculation.
INVPEND	7+	Inverted pendulum controller with noise.
PACK	10+	Packing objects with varying weights.
VOL	8+	Filling a tank with fluid at uncertain rates.
ALIGN-1,2	6+	Pointing to a target with error feedback.
CART	7+	Steering a cart against disturbances.

ID	N_s	N_u	T_s, T_{vc}	p_{mc}	$[lb, ub]$
EGFR EPI	563	45	0.1, 1.1	0.97803	[0.97803,0.97803]
ARTIRAL	19547	2520	15.6, 1095	0.94094	[0.82809,1.0]
CORONARY	7181	1239	3.9, 998.5	0.91992	[0.87239,1.0]
INVPEND	90	1	18.5, 0.1	1	[1,1]
PACK	8043	1010	4.6, 24.1	0.95052	[0.95051,0.95052]
ALIGN-1	4464	529	3.9, 25.7	0.90834	[0.90769,0.90846]
ALIGN-2	17892	1606	8.9 , 75.1	0.93110	[0.9304,0.9384]
VOL	121	9	3.6 , 1766	0.835	**[0,1]**
CART	5799	774	4.2 , 1612	0.94898	**[0.0176,1]**

Figure 5 Benchmarks and their evaluation[13]

InvPEND gives the highest Pmc and also it takes the highest simulation time. Pmc is known as monte-carlo probability estimate. INvPEND is basically a model of robot. VOL gives the lowest Pmc and it takes the lowest simulation time. But InvPEND has only 1 unique path and 90 simulated paths where VOL has 9 unique path and 121 simulated paths.

5 Abstract Interpretation: Discussion

All the 3 papers from abstract interpretation group talks about static analysis. AI^2 focuses on the safety and robustness of the neural network. They propose an application name AI2. They have shown that AI^2 is the more effective application for verifying the robustness of

the neural network. This paper has a major limitation. The researches have worked with MNIST and CIFAR but both have small data instances. Also, they have only shown the CNN and FN for their evaluation but what about the RNN? We do not know how the application would work on RNN? The probabilistic program paper presents a technique for the probabilistic program to estimate the probability by considering the path. The main weakness of this paper is that the researchers have found doubtable probability estimate for all benchmarks. For example, they have estimated 100% pmc for InvPEND, also all of the benchmarks give more than 91% pmc except one benchmark named VOL which estimates 83.5% pmc. Usually, in the statistics, this type of highest probability estimation is rarely seen. The OptOctagon paper reports the fastest static numeric program analysis. The research implements the extended version for APRON and the extended version is known as OptOctagon. The limitation of OptOctagon is that OptOctagon only works for the numerical domain. Already some researchers have introduced similar approaches in other domain [16][17].

6 Correctness

Soundness in the program analysis describes the ability to detect all possible errors [18]. Soundness is important in program analysis for error detection tools. A tool can be unsound regarding the effectiveness of the program. IRISA and INRIA researches have introduced a static analyzer named Veraso for most of the C program. Static analyzer can be used as a bug finder or a program verifier. They have proved the soundness of Verasco by coq proof assistant. Verasco has three layers such Abstract interpreter, state abstraction and numeric abstract domain. It translates the source of C into C# minor by using the CompCert compiler. C# minor is language in the completion pipeline of CompCert. CompCert compiler guarantees the soundness of the analysis. They have written 34000 lines of codes in the coq development for Verasco. The following figure shows the architecture of Verasco with specs, prrofs and percentage of overall code.

	Specs	Proofs	Overall
Interfaces	1081	139	3%
Abstract interpreter	2335	2204	13%
State abstraction	2473	3563	17%
Numerical domains	7805	8563	48%
Domain combinators	1305	1826	9%
Intervals	1489	2224	10%
Congruences	403	288	2%
Polyhedra (validator)	4038	3598	22% (from [19])
Symbolic equalities	570	627	3%
Miscellaneous libraries	3153	3329	19%
Total	16847	17040	33887 lines

Figure 6 the architecture of Verasco with specs, prrofs and percentage of overall code[26]

They have conducted an experiment with the analyzer. First, they have run the static analyzer in the C program that has approximate 100 lines of code, known as integr.c. The main goal is to check the run time errors. Then they have run the analyzer in smult.c, almabench.c and nbody.c. nbody.c and almabench.c perform the heavy numerical computation and sult.c

consists of scalar multiplication. The main specialty of Verasco is that it can verify the absence of run-time bugs. The figure describes the program with their size and run-time

Program	Size	Time[3]
`integr.c`	42 lines	0.1s
`smult.c`	330 lines	86.0s
`nbody.c`	179 lines	30.8s
`almabench.c`	352 lines	328.6s

Figure 7 Program's size and run-time[7]

This is the final theorem of Coq verification

Proof. `Theorem vanalysis_correct :`
forall prog res tr,
vanalysis prog = (res, nil) →
`program_behaves (semantics prog) (Goes_wrong tr)` →
False. ◀

The ICL researchers have tested the symbolic execution engines to find bugs in the concrete and symbolic execution component [19]. Symbolic execution is a testing method that determines inputs that cause each part of a program to run [20]. There are several symbolic execution engines such as KLEE, CREST, FuzzBALL , etc. The researchers have applied their test approach in this symbolic engines and they have found 20 different bugs. The test approach has 5 stages. With the help of Csmith, the researchers have generated random and deterministic program in the 1st stage named 'Generate Program'. In the 2nd stage, they have created and run the program into different versions such as native, single-path and multi-path. In the 3rd stage, the crash has been detected and the output has been found as well as made comparison. In the Gather Mismatches stage, the program that are mismatches between native and symbolic execution have been executed. Finally, these mismatches are reduced by C-Reducer and in the end, it has reported the bugs. The following table shows the bugs that are found from KLEE, CREST and FuzzBALL by applying the automatic testing.

Issue#	Bug description	Mode	Oracle	Reduced size (LOC)
246	**Some unions not retaining values**	C	output	11
247	Incorrect by value structure passing	C	output	18
747	Invalid overshift error triggered by optimisation bug in LLVM	C	output	5
163	**Vector instructions unhandled, caused by -O2 optimisations**	C	output	6
268	**Floating point exception**	C	crash	14
266	**Incorrect handling of division by 1**	SP	function calls & output	17
262	**Execution forks unexpectedly**	SP	function calls	14
261	**Segmentation fault due to % operator**	SP	crash	12
n/a[2]	**Incorrect casting from signed to unsigned short**	SP	output	27
308	Abnormal termination in STP solver	SP, MP	crash	10
n/a[1]	**Assertion failure in STP solver 1.6**	SP, MP	crash	–
264	Replaying read-only variables not handled	MP	crash	8
331	File system model and replay library interplay	MP	function calls	9
n/a[2]	**Divergence b/w test generating path and test replay path**	MP	function calls	21

[1] Not explored further as the bug seems to have been fixed in the newest release of STP.
[2] Fixed prior to reporting as the side effect of what looks to be an unrelated patch.

Issue#	Bug description	Mode	Oracle
CREST `github.com/jburnim/crest/issues/<Issue#>`			
7	Return struct errror	C	crash
6	**Big integer in expression**	SP	output
9	Non 32-bit wide bitfields	SP	output
FuzzBALL `github.com/bitblaze-fuzzball/fuzzball/issues/<Issue#>`			
21	**STP div by zero failure**[1]	SP	crash
20	**Strange term failure**	SP	crash
22	**Wrong behaviour**	SP	output

[1] Fixed in the upstream version of STP.

Figure 8 Bugs in KLEE, CREST and FuzzBALL[18]

The automatic testing has found the highest numbers of bugs in KLEE and same of bugs in CREST and FuzzBALL.

7 Correctness: Discussion

Verasco Papers describes the static analysis. This paper mainly focuses on the implementation of static analyzer named 'Verasco' with the coq proof assistance. Also, the paper shows a small experiment on different functionalities of programs to determine the run-time. The main limitation of the paper is that the research did not show any example for thousand lines of code. Based on some small program, they have declared that Verasco is effective for static program to find the soundness. On the other hand, Automatic paper present a testing approach to test the symbolic execution tools. This paper concerns the correctness of the symbolic execution tool. This paper represents the dynamic analysis though both Verasco and Automatic testing paper talks about the correctness. The weakness of the paper is that debugging large program is often hard under symbolic execution because large program consists of different types of variables. Also, there is some limitation happen when the researchers have tried to apply automatic testing into CREST symbolic execution tools because CREST does not support 64-bit integers.

8 Applications

Android applications are popular among this generation. Therefore, developing an application is most common in the software development world. Most of the Android application has rich features. So, Data race is a common term for this rich application. Some IIS researchers have implemented a tool named DroidRacer to detects data races in the android application. They

have also formalized the concurrent semantic and then they have introduced happens-before relation [21]. In happens-before relation explains the relation of two events. DroidRacer detects the data races in the android application with the help of happens-before relation. They have analyzed 15 application where 10 of them are open source. DroidRacer has 3 components. UI explorer concerns the UI related classes. Trace Generator involves the different types of log such as read, write, attach, etc. Race Detector constructs a happens-before graph and detects the races as well as classifies the races. The researchers have applied the DroidRaces in the following application to find the traces.

Application (LOC)	*Trace length*	*Fields*	*Threads (w/o Qs)*	*Threads (w/ Qs)*	*Async. tasks*
Aard Dictionary (4044)	1355	189	2	1	58
Music Player (11012)	5532	521	3	2	62
My Tracks (26146)	7305	573	11	7	164
Messenger (27593)	10106	845	11	4	99
Tomdroid Notes (3215)	10120	413	3	1	348
FBReader (50042)	10723	322	14	1	119
Browser (30874)	19062	963	13	4	103
OpenSudoku (6151)	24901	334	5	1	45
K-9 Mail (54119)	29662	1296	7	2	689
SGTPuzzles (2368)	38864	566	4	1	80
Remind Me	10348	348	3	1	176
Twitter	16975	1362	21	5	97
Adobe Reader	33866	1267	17	4	226
Facebook	52146	801	16	3	16
Flipkart	157539	2065	36	3	105

Figure 9 Application with Traces [20]

Open source applications have comparatively less thread than proprietary applications. In the proprietary section, Flipkart has the highest numbers of thread where 36 of them are without queues and 3 of them are with queues. Async task column represents the numbers of asynchronous calls. K-9 mail has most numbers of asynchronous call where Flipkart has only 105. The following table shows the reported data races.

Application	*Multi-threaded*	*Single-threaded*		
		Cross-posted	*Co-enabled*	*Delayed*
Aard Dictionary	1 (1)	0	0	0
Music Player	0	17 (4)	11 (10)	4 (0)
My Tracks	1 (0)	2 (1)	1 (0)	0
Messenger	1 (1)	15 (5)	4 (3)	2 (2)
Tomdroid Notes	0	5 (2)	1 (0)	0
FBReader	1 (0)	22 (22)	14 (4)	0
Browser	2 (1)	64 (2)	0	0
OpenSudoku	1 (0)	1 (0)	0	0
K-9 Mail	9 (2)	0	1 (0)	0
SGTPuzzles	11 (10)	21 (8)	0	0
Total	27 (15)	147 (44)	32 (17)	6 (2)
Remind Me	0	21	33	0
Twitter	0	20	7	4
Adobe Reader	34	73	0	9
Facebook	12	10	0	0
Flipkart	12	152	84	30
Total	58	276	124	43

Figure 10 Data Races by DroidRacer[20]

37% data races have been confirmed true positive from the open source application. In the proprietary section, a total of 546 data races have been found from these applications with

45 unknown categories. It is not possible to distinguish the TP/FP rate from the proprietary applications because the source codes are unknown. Also, some researchers from Facebook present another race detection tool named RacerD [22]. It is a static program analysis tool. It is developed at Facebook. It has the most important features such as speed, it can find issues quickly than any other race detection tools. Also, it can run on a large program. RaceD do not do analysis on the whole program and identify bug faster than DroidRacer. RacerD has two part. An algorithm that reports data races and specific abstract domain that computes the information. Another part is implementing the domain and computing the summaries. RacerD is implemented with INFER.AI analysis framework. RacerD is used for two reason at Facebook.

- Preventing regressions in safe concurrent code [22]
- Adapting sequential code for safe use in a concurrent context [22]

The following figure shows the comparison between RacerD and DroidRacer

Program	# Files	# LOC	# Alarms (true bugs)		RacerD Runtime
			DroidRacer	RacerD	
SGTPuzzles	33	9,459	11 (10)	18 (18)	12s
OpenSudoku	62	9,021	1 (0)	14 (14)	14s
K-9 mail	3303	78,503	2 (1)	185 (>3)	2m 33s

Figure 11 RacerD vs DroidRacer [21]

RacerD has found more errors than DroidRacer as well as RacerD provides 100% TP rate. A group of researchers has implemented a static taint analysis named FlowDroid to reduce the number of false alarm. In the current work, data leakage is one of the common phenomena. Therefore detecting this type of data leakage is necessary. FlowDroid detects data leaks. It has 4 sensitivity such as Context, Flow, Field and Object. The researchers have also proposed DrodBench tool to evaluate the accuracy and the effectiveness of static analysis tools. There is already a similar framework for java program analysis named Soot framework. FlowDroid is the extended version of the Soot framework. The researchers did some experiment on commercial taint-analysis tools, InsecureBank, Real World application and SecuriBench Micro. They have compared the FlowDroid with the help of DroidBench. They have made an evaluation on 39 hand craft applications and have found 93% recall and 56% precision. It only takes 31 second to analyze the InsecureBank and FlowDroid has found 7 data leaks. They researchers have also applied FlowDroid on real world application. They have used FlowDroid into standford SecuriBench Micro and found the following test results.

Test-case group	TP	FP
Aliasing	11/11	0
Arrays	9/9	6
Basic	58/60	0
Collections	14/14	3
Datastructure	5/5	0
Factory	3/3	0
Inter	14/16	0
Pred	n/a	n/a
Reflection	n/a	n/a
Sanitizer	n/a	n/a
Session	3/3	0
StrongUpdates	0/0	0
Sum	117/121	9

Figure 12 Test Results [22]

TP represents the actual leaks and for Datastructure, they have found 5 out 5 but they have not found anything for the StrongUpdates. FlowDroid is a highly precise tool. It provides a better result than other static taint-analyis tools such as SCanDroid, Leak-Miner, etc.

9 Applications: Discussion

RacerD and FlowDroid are static analysis tool and on the other side, DroidRacer is dynami analysis tool. RacerD and DroidRacer both concern the data detection. The comparison result says RacerD is better than DroidRacer because RacerD finds more bugs than DroidRacer. Also, RacerD is developed by a service company. Therefore, RacerD has been tested more than DroidRacer. DroidRacer has few limitations such as it provides some data races that are in unknown categories. Also, it can not provide the TP and FP for a large and proprietary application. But, RacerD provides data races with the known category. RaceD has also some limitation such as weak memory, misuse of subtler forms, etc. RacerD sometimes misses races because of aliasing. FlowDroid presents a way to find data leakage and reduce false alarm. The problem of FlowDroid is that it cannot resolves reflective calls if the arguments are not string-constant in the Java platform.

10 Beyond

Model checking is important to verify the correctness of a model and it's an undecidable program. In the model checking, the program can be verified or not verified or the model checker can be failed. A group of researchers has shown conditional model checkers that do not fail but in the time of time-out, model checkers summarize their work [24]. Conditional model checker gives the maximum outcome than other general model checkers. Conditional model checker only checks those part of the model that are not meet the expectation of the condition. Conditional takes those part as inputs and then it outputs a condition that summarizes those parts. Some researchers from EPFL have presented an approach for varying resource [25]. They have used the approaches on 17 data structures and algorithm. The benchmarks are lazy mergesort, Okasaki's real-time queue and deque and scale code. The researchers have shown the contract-based approach. They have specified the resource bounds for a program. There are two parts of their approaches. First, generate the first order program, then verify the contracts of the first order program. They have proved the resource bound of 17 benchmarks by implementing lazy data structure. After that, they have evaluated the accuracy of the bound and have found 80

11 Beyond: Discussion

Condition model checking paper satisfy the model checkers that are usually failed. This is a dynamic analysis. On the other side, another paper concerns resource verification. This is static analysis. There is no specific similarity between these two papers except the program analysis part. One paper shows a new way of model checking where another paper concerns the proving part. The limitation of conditional model checker is that every model checker is undecidable whether it is conditional or not. In the resource paper, the approach can tackle many complex programs. Also, the approach easily works on a large program.

12 Conclusion and Discussion

The report illustrates the 5 dynamic analysis papers and 7 static analysis papers. Among all of the papers, RacerD paper is the most practical and interesting paper as the application is developed by Facebook to detect data races and also it outperforms the DroidRacer paper. RacerD overcomes the limitation of DroidRacer as DroidRacer finds data races with unknown category and provides little true positive. RacerD has no obligation to detect data races and also, it achieves 100% TP rate. Another interesting researcher paper is AI2 as right now, the neural network is the most demanding statistical learning technique. We can implement a neural network in almost every fields of computer science. The least interesting papers are conditional model checking paper and Static Probability paper. Model checking is always undecidable, so it does not matter whether it conditional or non-conditional. The problem with the Static Probability paper is that it achieves high probability for every benchmark which does not ease for modern statistical learning. Modern statistical learning trains a large amount of data, therefore it is almost difficult to achieve high probabilities between 90-100. All the paper show the good applications for program analysis and these applications have a great impact in the field of program analysis.

References

References

1 Program Analysis
https://en.wikipedia.org/wiki/Program_analysis

2 Böhme, Marcel, et al. "Directed greybox fuzzing." Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. ACM, 2017.

3 Simulated Annealing
https://www.mathworks.com/help/gads/what-is-simulated-annealing.html

4 AFLGo
https://github.com/aflgo/aflgo

5 OSS-Fuzz
https://github.com/aflgo/oss-fuzz

6 KATCH
https://srg.doc.ic.ac.uk/projects/katch/

7 Böhme, Marcel, et al. "Driller: Augmenting Fuzzing Through Selective Symbolic Execution." NDSS. Vol. 16. No. 2016. 2016.

8 Concolic Execution
http://www.verimag.imag.fr/m̃ounier/Enseignement/Software_Security/ConcolicExecution.pdf

9 Abstract Interpretation
http://pages.cs.wisc.edu/h̃orwitz/CS704-NOTES/ABSTRACT-INTERPRETATION.html

10 Abstract interpretation
https://en.wikipedia.org/wiki/Abstract_interpretation

11 Gehr, Timon, et al. "Ai2: Safety and robustness certification of neural networks with abstract interpretation." 2018 IEEE Symposium on Security and Privacy (SP). IEEE, 2018.

12 Singh, Gagandeep, Markus Püschel, and Martin Vechev. "Making numerical program analysis fast." ACM SIGPLAN Notices. Vol. 50. No. 6. ACM, 2015.

13 The Octagon Abstract Domain
https://www-apr.lip6.fr/ mine/publi/article-mine-HOSC06.pdf

14 Sankaranarayanan, Sriram, Aleksandar Chakarov, and Sumit Gulwani. "Static analysis for probabilistic programs: inferring whole program properties from finitely many paths." ACM SIGPLAN Notices. Vol. 48. No. 6. ACM, 2013.

15 Probabilistic Analysis
`https://sites.google.com/site/probabilisticAnalysis`

16 R. Claris and J. Cortadella. The octahedron abstract domain. In Proc. International Static Analysis Symposium (SAS), volume 3148 of Lecture Notes in Computer Science, pages 312–327. Springer, 2004.

17 P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In Proc. ACM Symposium on Principles of Programming Languages (POPL), pages 84–96, 1978.

18 The Octagon Abstract Domain
`https://www.cs.umd.edu/ pugh/BugWorkshop05/papers/12-xie.pdf`

19 Kapus, Timotej, and Cristian Cadar. "Automatic testing of symbolic execution engines via program generation and differential testing." Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering. IEEE Press, 2017.

20 The Octagon Abstract Domain
`https://en.wikipedia.org/wiki/Symbolic_execution`

21 Maiya, Pallavi, Aditya Kanade, and Rupak Majumdar. "Race detection for Android applications." ACM SIGPLAN Notices. Vol. 49. No. 6. ACM, 2014.

22 Blackshear, Sam, et al. "RacerD: compositional static race detection." Proceedings of the ACM on Programming Languages 2.OOPSLA (2018): 144.

23 Arzt, Steven, et al. " Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps." Acm Sigplan Notices 49.6 (2014): 259-269.

24 Beyer, Dirk, et al. "Conditional model checking: a technique to pass information between verifiers." Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering. ACM, 2012.

25 Madhavan, Ravichandhran, Sumith Kulal, and Viktor Kuncak. "Contract-based resource verification for higher-order functions with memoization." Acm Sigplan Notices. Vol. 52. No. 1. ACM, 2017.

26 Jourdan, Jacques-Henri, et al. "A formally-verified C static analyzer." ACM SIGPLAN Notices 50.1 (2015): 247-259.