Poking the Bear: Lessons Learned from Probing Three Android Malware Datasets

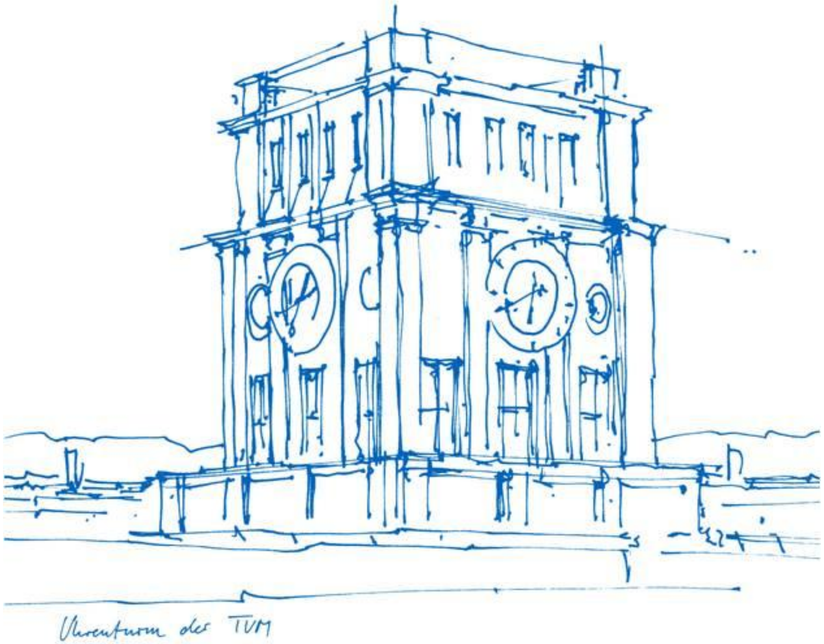
Aleieldin Salem and Alexander Pretschner

Technische Universität München

Garching bei München

{salem, pretschn @in.tum.de}

Montpellier, 04.09.2018



Abstract

- Stumbled upon some inconsistencies while experimenting with different Android malware datasets
- Investigate the source of discrepancies
- A series of experiments performed on three Android malware datasets
- Some (interesting) findings

Background

- Working on a solution based on “Active Learning”

- Evaluating on Malgenome vs. Piggybacking
  - Datasets of Repackaged/Piggybacked Malware
  - Malgenome = great results!
  - Piggybacking = mediocre results?

- Trying on AMD and Drebin
  - Works like a charm!

- What the .. ?

Research Questions

RQ1 : What are the trends adopted by Android malware authors according to the malicious apps in current datasets (i.e., malware families/types, app marketplaces, distribution techniques, etc.)? And how did they evolve over the years?

RQ2 : What is the lifespan of a malware dataset within which it can be used to train effective detection methods?

RQ3 : How do conventional detection methods (e.g., machine learning classifiers), fare against different malware datasets?

RQ4 : What are the malware families/types that are most difficult to detect, if any?

RQ5 : How can malware authors circumvent effective detection methods?

Dissection Experiments

- Infer some information about the malicious instances found in:
  - Malgenome (Zhou et al. 2012)
  - Piggybacking (Li et al. 2017)
  - AMD (Wei et al. 2017)

- VirusTotal detection rates, involved marketplaces, malware types, etc.

- Backed up by information in Euphony (Hurier et al. 2017)

Dissection Experiments

- Backed up by information in Euphony (Hurier et al. 2017)

around 50

Dataset	Total Apps	Average # of VT Detectors	Source(s)	Top Families	Top Types
Malgenome (2010-2012)	1234	31.43	Official + Alternative Markets [5]	Droidkungfu (38%) Basebridge (25%) Geinimi (5%)	Trojan (94%) Exploit (3%) Spyware (1%)
Piggybacking (Malicious apps) (2016)	1136	≈ 9	Anzhi (64%) Appchina (12%) Angeeks (5%)	Dowgin (24%) Kuguo (22%) Gingermaster (6%)	Adware (64%) Trojan (25%) Spyware (2%)
AMD (2010-2016)	204 (of 1250)	24.6	Malgenome (29%) Google Play (27%) Appchina (23%)	Droidkungfu (20%) Airpush (8%) Ginmaster (8%)	Trojan (42%) Adware (34%) Exploit (11%)

More information: https://androidmalwareinsights.github.io

Dissection Experiments

- Backed up by information in Euphony (Hurier et al. 2017)

around 50

Dataset	Total Apps	Average # of VT Detectors	Source(s)	Top Families	Top Types
Malgenome (2010-2012)	1234	31.43	Official + Alternative Markets [5]	Droidkungfu (38%) Basebridge (25%) Geinimi (5%)	Trojan (94%) Exploit (3%) Spyware (1%)
Piggybacking (Malicious apps) (2016)	1136	$\boxed{\approx 9}$	Anzhi (64%) Appchina (12%) Angeeks (5%)	Dowgin (24%) Kuguo (22%) Gingermaster (6%)	Adware (64%) Trojan (25%) Spyware (2%)
AMD (2010-2016)	204 (of 1250)	24.6	Malgenome (29%) Google Play (27%) Appchina (23%)	Droidkungfu (20%) Airpush (8%) Ginmaster (8%)	Trojan (42%) Adware (34%) Exploit (11%)

More information: https://androidmalwareinsights.github.io

Dissection Experiments

- Backed up by information in Euphony (Hurier et al. 2017)

Dataset	Total Apps	Average # of VT Detectors	Source(s)	Top Families	Top Types
Malgenome (2010-2012)	1234	31.43	Official + Alternative Markets [5]	Droidkungfu (38%) Basebridge (25%) Geinimi (5%)	Trojan (94%) Exploit (3%) Spyware (1%)
Piggybacking (Malicious apps) (2016)	1136	≈ 9	Anzhi (64%) Appchina (12%) Angeeks (5%)	Dowgin (24%) Kuguo (22%) Gingermaster (6%)	Adware (64%) Trojan (25%) Spyware (2%)
AMD (2010-2016)	204 (of 1250)	24.6	Malgenome (29%) Google Play (27%) Appchina (23%)	Droidkungfu (20%) Airpush (8%) Ginmaster (8%)	Trojan (42%) Adware (34%) Exploit (11%)

More information: https://androidmalwareinsights.github.io

Dissection Experiments (cont'd)

- What about repackaging?

- What is in fact the definition of repackaging?
  - E.g. must the app be decompiled/disassembled?

- Wei et al. [authors of AMD] claim it has been declining

- How to quickly infer whether an app is repackaged?

- Simple technique using compiler fingerprinting (with APKiD[1])

[1] https://rednaga.io/2016/07/31/detecting_pirated_and_malicious_android_apps_with_apkid/

Dissection Experiments (cont'd)

- Simple technique using compiler fingerprinting (with APKiD[1])
- Legitimate developer = access to source code = using IDE
- Compile app using Android SDK's `dx` and `dexmerge` compilers
- If app compiled using other compilers (e.g., `dexlib`)
  = repackaged = no access to source code != legitimate developer?
- Different compilers leave unique marks on the compiled code

[1] https://rednaga.io/2016/07/31/detecting_pirated_and_malicious_android_apps_with_apkid/

Dissection Experiments (cont'd)

- What about repackaging?

- What is in fact the definition of repackaging?

Dataset	dx	dexmerge	Not repackaged (dx + dexmerge)	Repackaged (dexlib 1.X + 2.X)
Malgenome	52%	–	52%	48%
Play Store (benign)	61%	34%	95%	5%
Piggybacking (malicious)	22%	6%	28%	72%
Piggybacking (benign)	61%	22%	73%	17%
AMD	38%	35%	63%	≈27%

Dissection Experiments (cont'd)

- What about repackaging?

- What is in fact the definition of repackaging?

Dataset	dx	dexmerge	Not repackaged (dx + dexmerge)	Repackaged (dexlib 1.X + 2.X)
Malgenome	52%	–	52%	48%
Play Store (benign)	61%	34%	95%	5%
Piggybacking (malicious)	22%	6%	28%	72%
Piggybacking (benign)	61%	22%	73%	17%
AMD	38%	35%	63%	≈27%

lazy developers?

wrong labeling?

Dissection Experiments (cont'd)

- What about repackaging?
- What is in fact the definition of repackaging?

86% repackaged?!

Dataset	dx	dexmerge	Not repackaged (dx + dexmerge)	Repackaged (dexlib 1.X + 2.X)
Malgenome	52%	–	52%	48%
Play Store (benign)	61%	34%	95%	5%
Piggybacking (malicious)	22%	6%	28%	72%
Piggybacking (benign)	61%	22%	73%	17%
AMD	38%	35%	63%	≈27%

declining?

Detection Experiments

- How do conventional detection techniques fare against different datasets?

- Conventional:
  - Machine learning classifiers
  - Trained with static/dynamic features
  - Validated using K-fold CV

Detection Experiments

- How do <u>conventional</u> detection techniques fare against different datasets?

- Ensemble classifier
    - KNN, with K = {10, 25, 50, 100, 250, 500}
    - Random Forests with estimators = {10, 25, 50, 75, 100}
    - Support Vector machine with linear kernel
    - 10-Fold CV

- Trained with static/dynamic features
    - <u>Static</u>: Extracted from APK using androguard
    - <u>Dynamic</u>: Running apps within VM + recording issued API calls

Detection Experiments

- How do conventional detection techniques fare against different datasets?

Dataset	Accuracy		Recall		Precision		Specificity		F1 Score	
	Static	Dynamic	Static	Dynamic	Static	Dynamic	Static	Dynamic	Static	Dynamic
Malgenome+GPlay	0.98	0.94	0.97	0.94	0.99	0.74	0.99	0.94	0.98	0.83
Piggybacking	0.67	0.67	0.70	0.70	0.63	0.76	0.65	0.61	0.63	0.73
AMD+GPlay	0.94	0.87	0.92	0.87	0.96	0.85	0.96	0.87	0.94	0.86

Detection Experiments

- How do conventional detection techniques fare against different datasets?

Dataset	Accuracy		Recall		Precision		Specificity		F1 Score	
	Static	Dynamic	Static	Dynamic	Static	Dynamic	Static	Dynamic	Static	Dynamic
Malgenome+GPlay	0.98	0.94	0.97	0.94	0.99	0.74	0.99	0.94	0.98	0.83
Piggybacking	0.67	0.67	0.70	0.70	0.63	0.76	0.65	0.61	0.63	0.73
AMD+GPlay	0.94	0.87	0.92	0.87	0.96	0.85	0.96	0.87	0.94	0.86

- But why?
  - Piggybacking = original, benign apps + repackaged, malicious versions
  - Majority = Adware
  - ~70% of misclassified apps = Adware

Detection Experiments (cont'd)

- What is the lifespan of malware datasets?
- Can we use an old/new dataset to detect newer/older datasets?
- Train voting classifier using dataset A, and test using dataset B

Detection Experiments (cont'd)

- What is the lifespan of malware datasets?
- Can we use an old/new dataset to detect newer/older datasets?
- Train voting classifier using dataset A, and test using dataset B

Training Dataset	Test Dataset	Accuracy		Recall		Precision		Specificity		F1 Score	
		Static	Dynamic	Static	Dynamic	Static	Dynamic	Static	Dynamic	Static	Dynamic
Malgenome+GPlay	Piggybacking	0.49	0.52	0.49	0.65	0.54	0.42	0.47	0.44	0.51	0.51
Malgenome+GPlay	AMD+GPlay	0.90	0.79	0.96	0.93	0.83	0.60	0.86	0.73	0.90	0.73
AMD+GPlay	Piggybacking	0.50	0.59	0.50	0.63	0.75	0.73	0.48	0.78	0.60	0.69
Piggybacking	AMD+GPlay	0.47	0.63	0.47	0.57	0.48	0.86	0.48	0.78	0.47	0.69
AMD+GPlay	Malgenome+GPlay	0.97	0.93	0.95	0.92	0.99	0.93	0.99	0.94	0.97	0.92
Piggybacking	Malgenome+GPlay	0.51	0.63	0.51	0.55	0.34	0.94	0.51	0.89	0.40	0.70

Adversarial Experiments

- How can an adversary make use of this?
- Consider a marketplace using a ML classifier as its “bouncer”
- The classifier is trained using malicious + benign apps
- If I [adversary] figure out one (or more) of the benign apps
- Repackage benign apps + upload to marketplace
- Classifier will be confused!!

Adversarial Experiments (cont'd)

- How can an adversary make use of this?
- If I [adversary] figure out one (or more) of the benign apps
- Many people presume apps on Google Play to be benign
- Use Google Play apps as benchmark/reference for benign behaviors
- Adversary make the same assumption!

Adversarial Experiments (cont'd)

- Piggybacking dataset = benign apps + repackaged versions
- Train voting classifier with dataset A, and test with dataset B
- Observe the effect of adding “Original” segment of Piggybacking on classification accuracy

Adversarial Experiments

- Observe the effect of adding "Original" segment of Piggybacking on classification accuracy

#	Training Dataset	Test Dataset	Accuracy	
			Static	Dynamic
1	AMD+GPlay	Piggybacked	0.81	0.72
2	AMD+GPlay	Original	0.20	0.38
3	AMD+Original	Piggybacked	0.17	0.50
4	AMD+Original	Original	0.98	0.94
5	AMD+Malgenome+GPlay	Piggybacked	0.81	0.79
6	AMD+Malgenome+GPlay	Original	0.20	0.30
7	AMD+Original+GPlay	Piggybacked	0.19	0.34
8	AMD+Original+GPlay	Original	0.98	0.98
9	AMD+Malgenome+Original+GPlay	Piggybacked	0.30	0.43
10	AMD+Malgenome+Original+GPlay	Original	0.91	0.92

Adversarial Experiments

- Observe the effect of adding “Original” segment of Piggybacking on classification accuracy

#	Training Dataset	Test Dataset	Accuracy	
			Static	Dynamic
1	AMD+GPlay	Piggybacked	0.81	0.72
2	AMD+GPlay	Original	0.20	0.38
3	AMD+Original	Piggybacked	0.17	0.50
4	AMD+Original	Original	0.98	0.94
5	AMD+Malgenome+GPlay	Piggybacked	0.81	0.79
6	AMD+Malgenome+GPlay	Original	0.20	0.30
7	AMD+Original+GPlay	Piggybacked	0.19	0.34
8	AMD+Original+GPlay	Original	0.98	0.98
9	AMD+Malgenome+Original+GPlay	Piggybacked	0.30	0.43
10	AMD+Malgenome+Original+GPlay	Original	0.91	0.92

Conclusion

RQ1 : What are the trends adopted by Android malware authors according to the malicious apps in current datasets (i.e., malware families/types, app marketplaces, distribution techniques, etc.)? And how did they evolve over the years?

- Trojans appear to be most popular malware type
- Adware is the go-to model for repackaging
- Repackaging is losing popularity
- Malicious apps continue to bypass Google Play’s safeguards

Conclusion (cont'd)

RQ2 : What is the lifespan of a malware dataset within which it can be used to train effective detection methods?

- AMD is 5-6 years younger than Malgenome
- Yet, apps from Malgenome are still out there!
- Malware authors prefer re-using/building on older malware
- Five years to use a dataset for training?

Conclusion (cont'd)

RQ3 : How do conventional detection methods (e.g., machine learning classifiers), fare against different malware datasets?

RQ4 : What are the malware families/types that are most difficult to detect, if any?

- Already answered that in the detection experiments.
- <u>Adware</u> most challenging to detect = Ambiguous nature
- Binary-labeling problem? What are the alternatives?

Conclusion (cont'd)

RQ5 : How can malware authors circumvent effective detection methods?

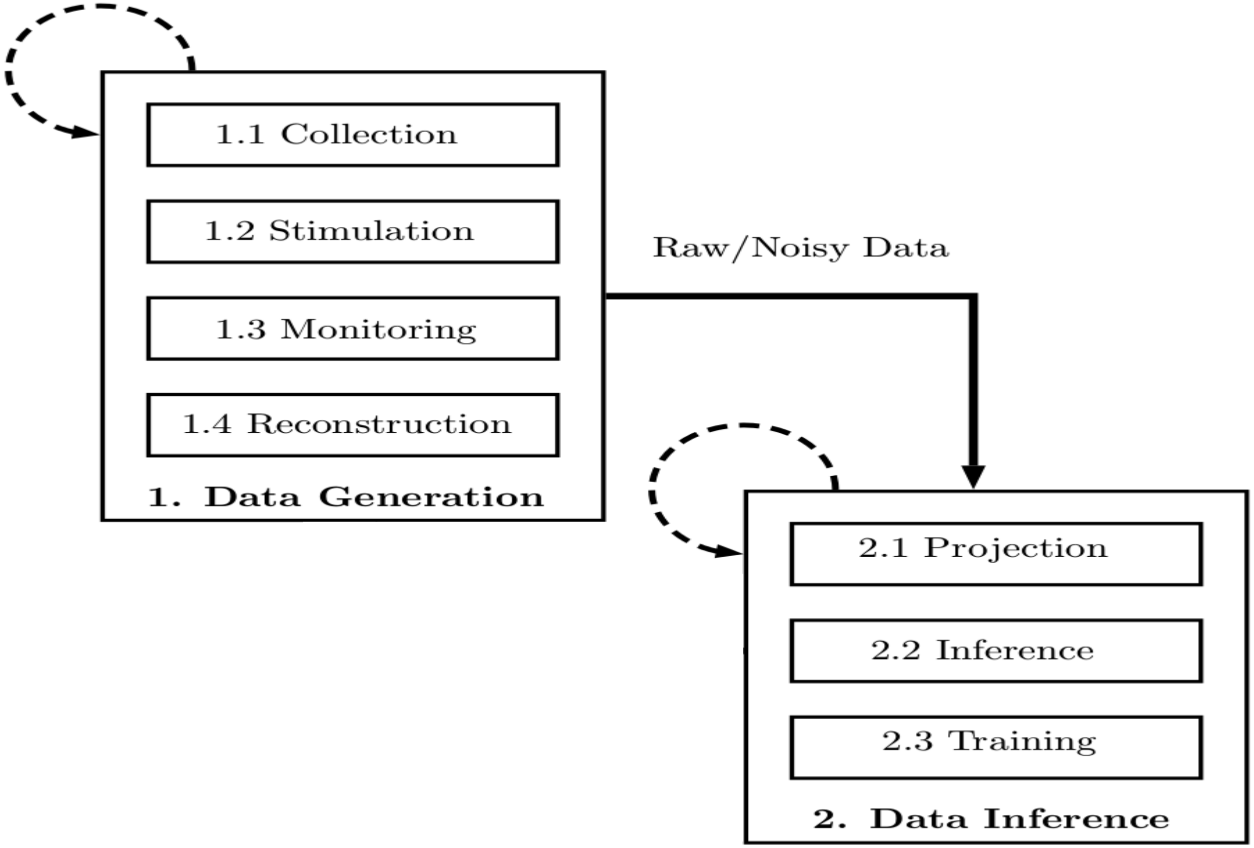
- In what we called as “adversarial setting”
- Effectively circumvent app vetting safeguards (especially ML-based ones)
- Repackaging benign apps used during training

Thank You

Any questions?

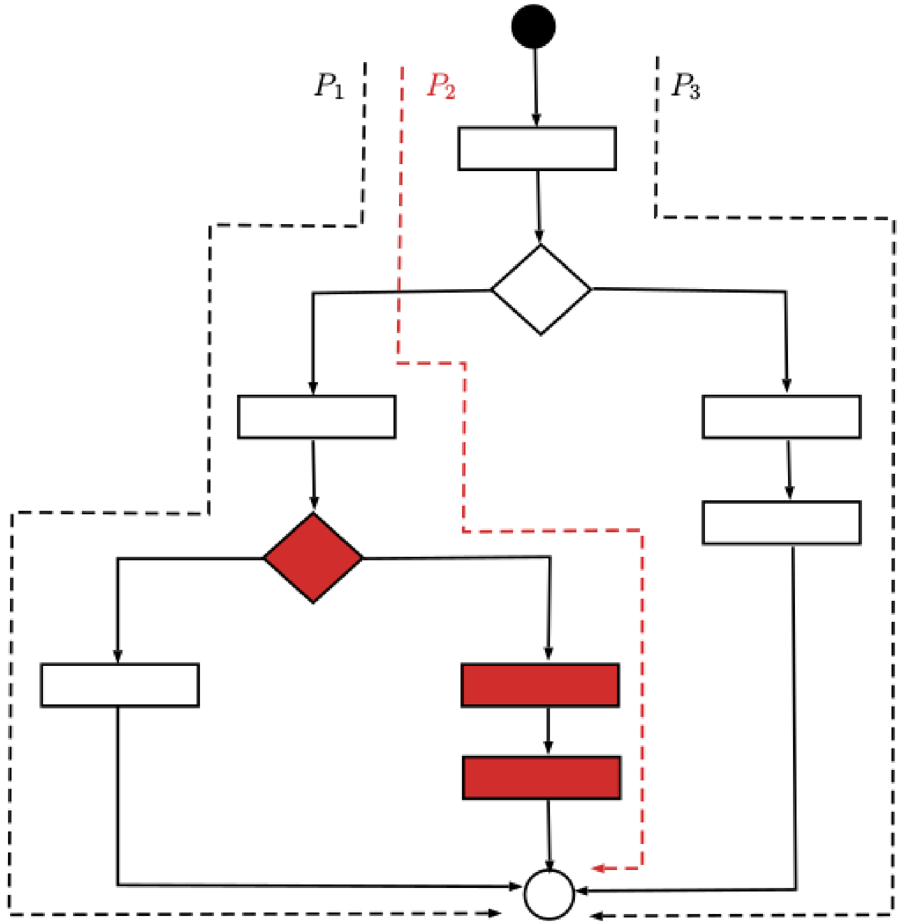
How it all began

- Working on a solution based on “Active Learning”



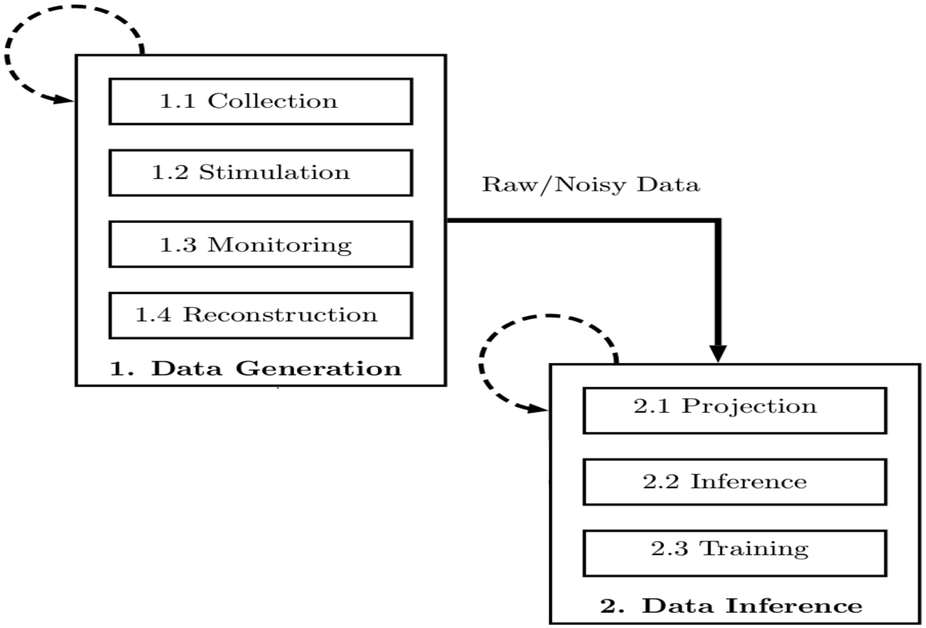
How it all began

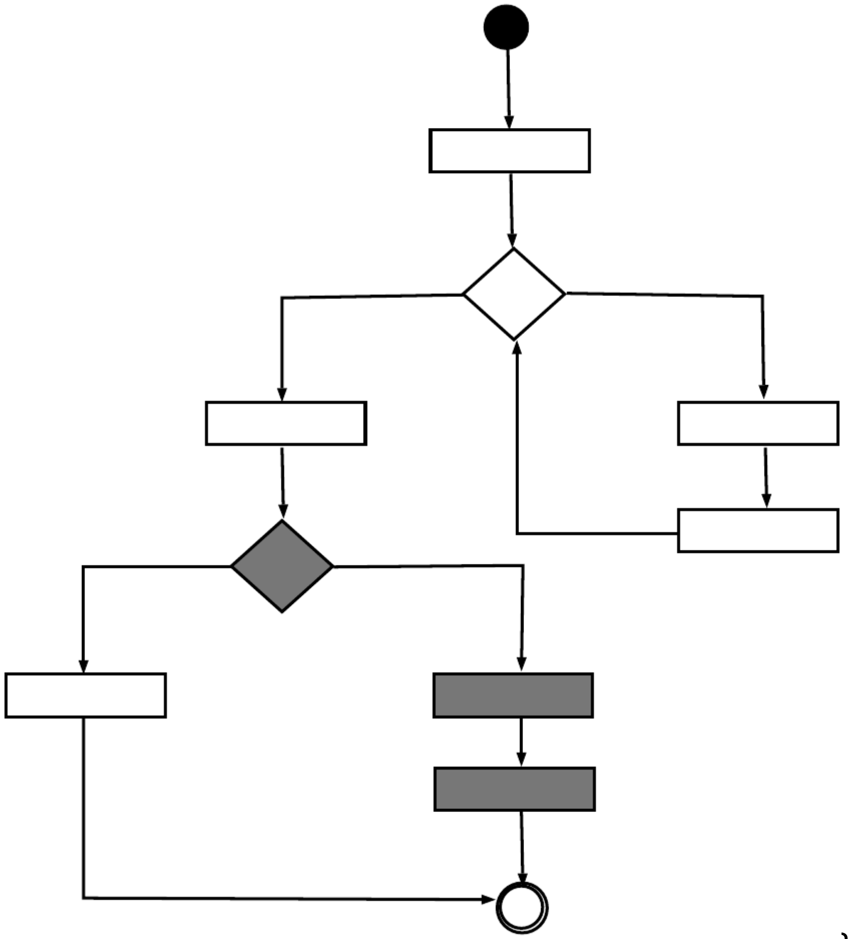
- Working on a solution based on “Active Learning”



How it all began

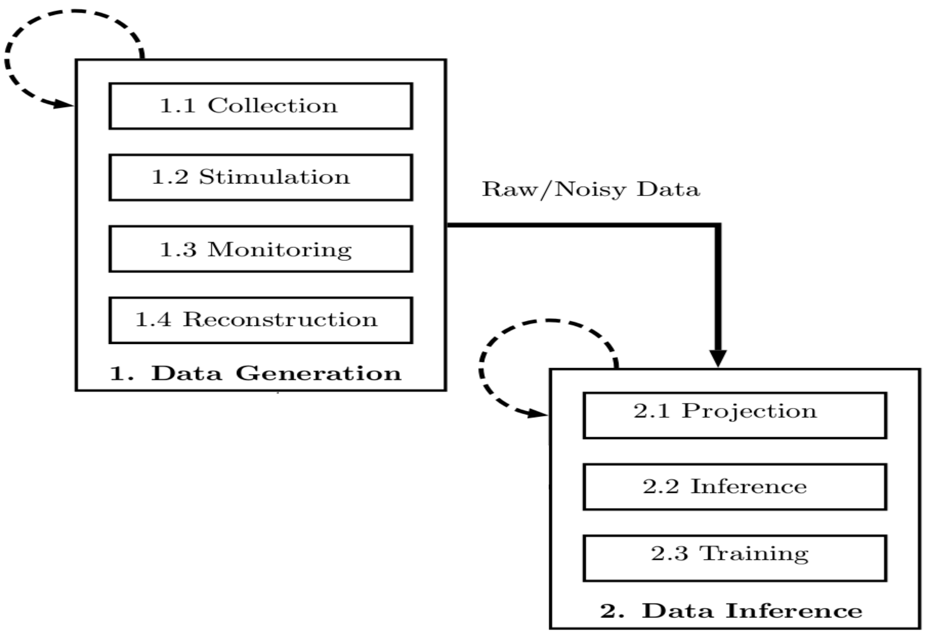
- Working on a solution based on “Active Learning”

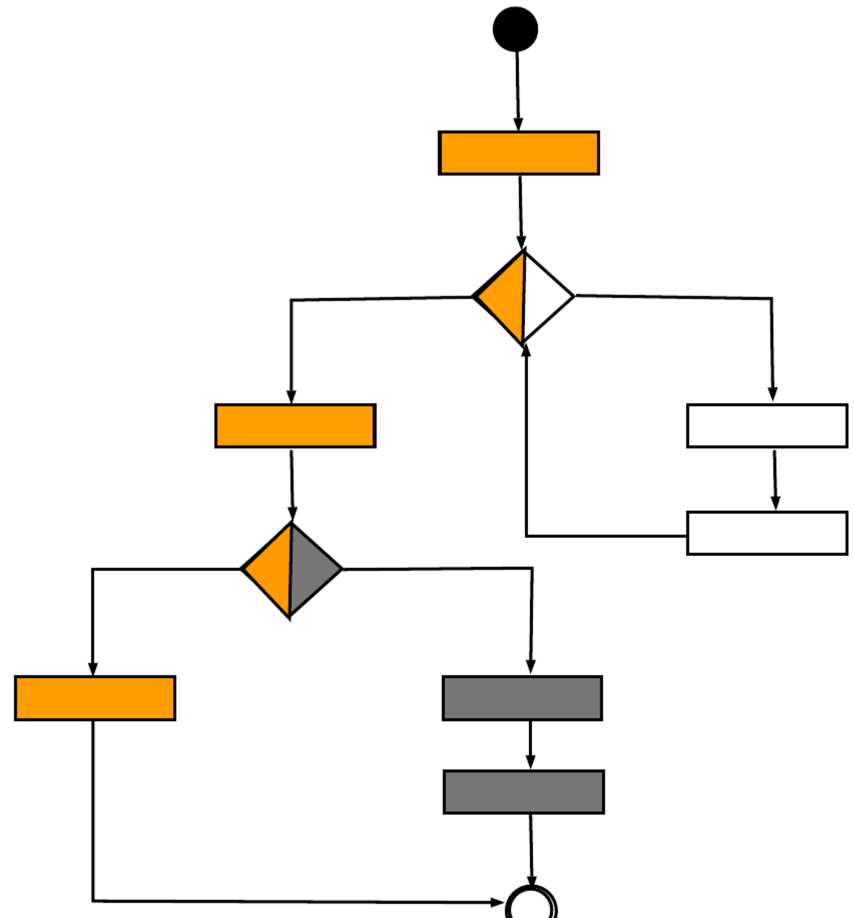




How it all began

- Working on a solution based on "Active Learning"





How it all began

- Working on a solution based on “Active Learning”