Philipps Universität Marburg

AndroLyzeLab

[A scalable android application package analyzer]

Nils Schmidt

Overview

1. Intro
2. Related work
3. General overview of AndroLyzeLab
4. Typical workflow
5. Scripts
6. Storage
7. Experiments
8. Future work

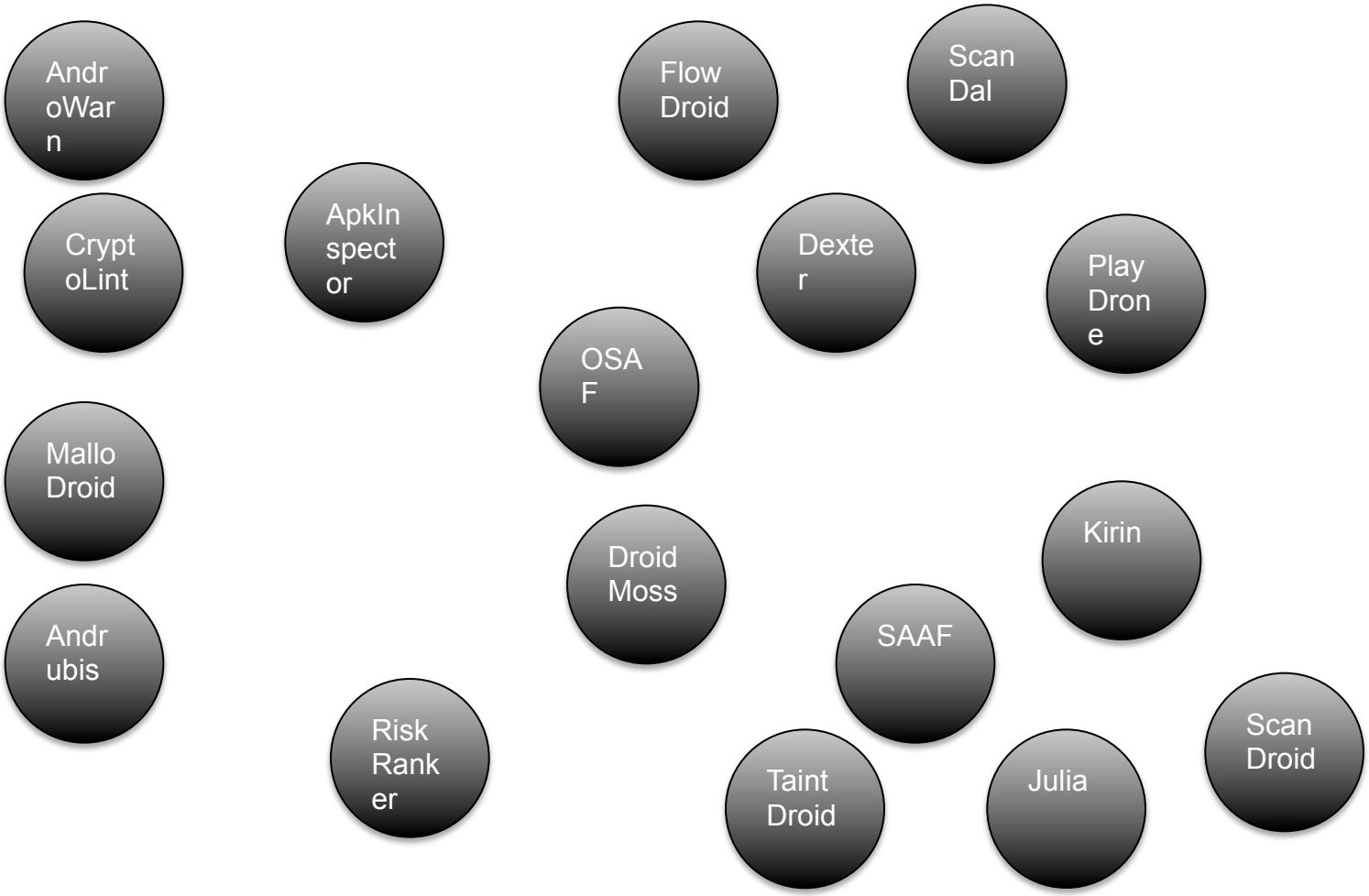
Intro

- Usage of mobile devices ➔ sensitive data
  - Address book, mail, chats, pictures, documents, metadata

- Can we trust applications?
  - Information leakage
  - Malware

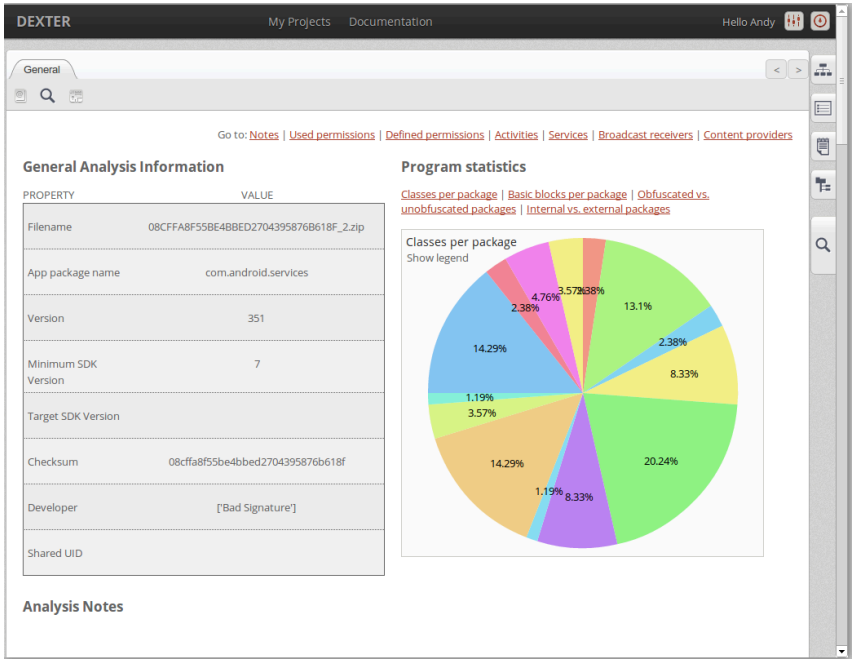
- Android:
  - market share: 78,4 % (2nd quarter 2013)
  - 900 million devices activated since 2008
  - +1,5 million per day
  - Sophos: exponential growth in malware

Related work

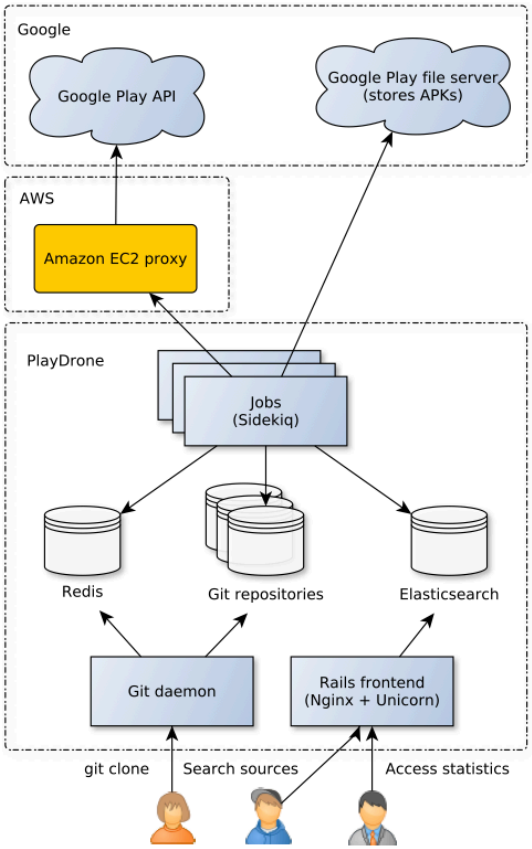


Related work: Dexter

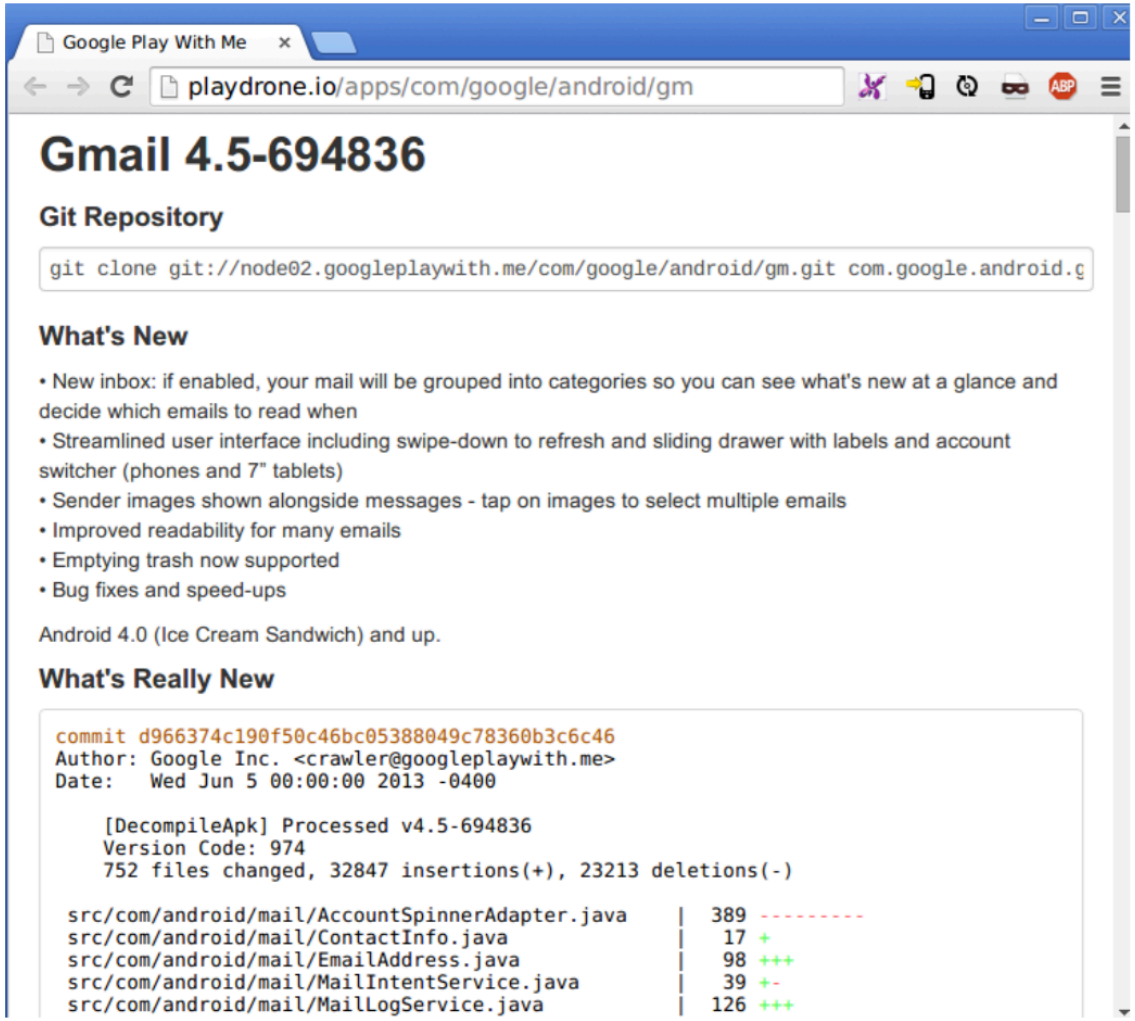
- Static android analysis tool
- Closed source, web platform
- Disassembly, decompilation
- General information and statistics
- Permissions
- UML-class diagrams
- Search
- Tagging-system
- Share results with other users
- Scalable?



Related work: PlayDrone



Related work: PlayDrone



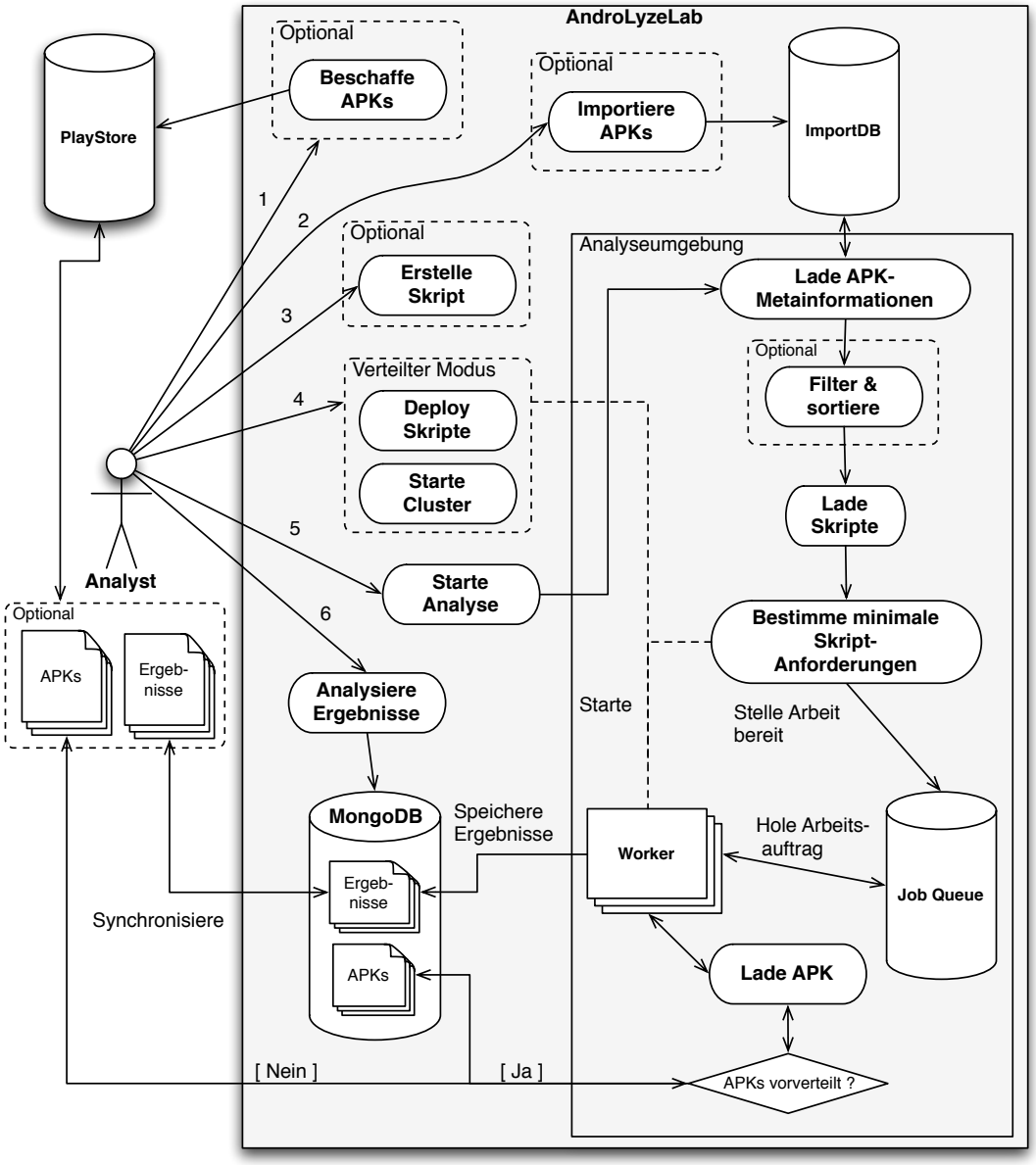
Overview

- Main purpose?
  - Analyze APKs
- How?
  - Use analysis functionality of androguard
  - Includes disassembler and ***native*** decompiler (DAD)
- From where do we get them?
  - Use *PlaystoreCrawler* to download APKs from Google Play Store
  - Update whole APK collection
- How do we organize many APKs?
  - Import metadata into database (sqlite3)
  - Adds filtering capabilities -> Only analyze subset matching filter
- How do we store analysis results?
  - Use MongoDB
  - Schema free database (NoSQL) for dynamic results
  - Filesystem storage optional

Overview #2

- How to extend it?
  - Use AndroLyzeLab script framework (python2)
  - Think about a result structure
  - Analyze with androguard
  - Use logging system to store results

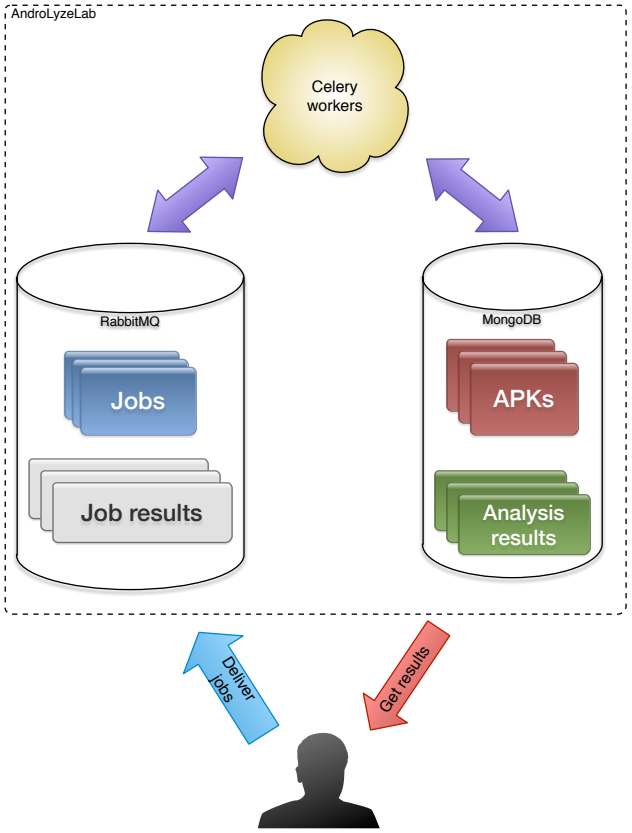
- How to evaluate results?
  - Query MongoDB
  - Use integrated query builder
  - Complex: MapReduce & JavaScript


AndroLyzeLab
Optional
Beschaffe APKs
PlayStore
Optional
Importiere APKs
ImportDB
1
2
Optional
Erstelle Skript
3
Analyseumgebung
Lade APK-Metainformationen
Optional
Filter & sortiere
Verteilter Modus
4
Deploy Skripte
Starte Cluster
Lade Skripte
5
Starte Analyse
Analyst
6
Optional
APKs
Ergeb-nisse
Analysiere Ergebnisse
Bestimme minimale Skript-Anforderungen
Starte
Stelle Arbeit bereit
MongoDB
Speichere Ergebnisse
Worker
Hole Arbeits-auftrag
Job Queue
Ergeb-nisse
APKs
Synchronisiere
Lade APK
[Nein]
[Ja]
APKs vorverteilt ?

Parallelization

- Determine minimum script requirements
- Open APK with androguard
- Run script(s)
- Store result(s) in mongoDB
- Use processes due to CPython Global Interpreter Lock
- Granularity
  - 1 Process = APK with all scripts
  - Due to androguard APK open overhead
- Local
  - Use producer-worker pattern with queues

Parallelization: Distributed



- Message-oriented middleware
- Publish jobs in queue
  - Serialize APK
  - Or APK ID (for mongoDB)
- Store results in mongoDB
- Store result IDs in result queue
- Fetch results from mongoDB

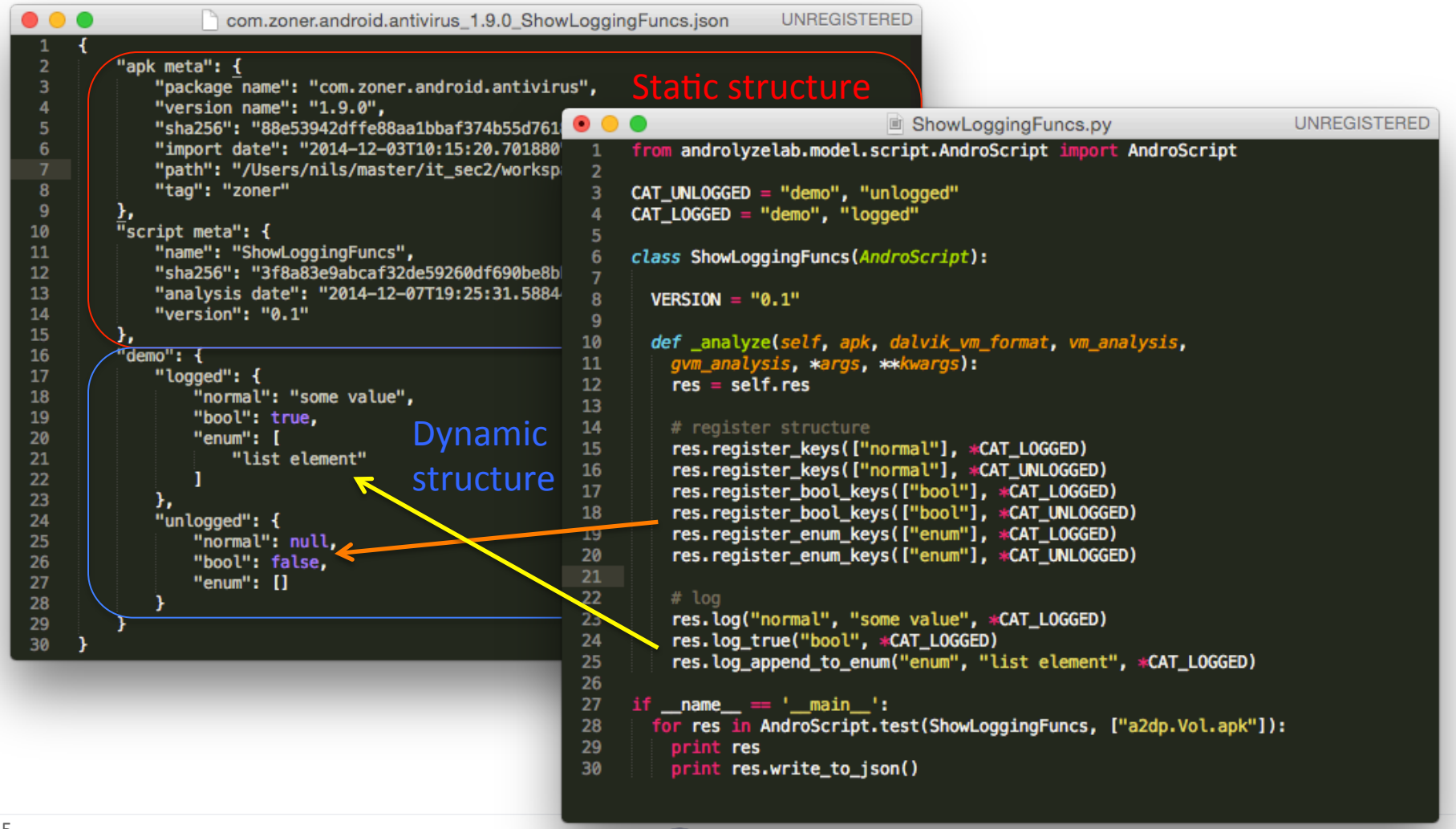
Design: Message format

Argument	Type
Scripts	List<String> // package name
Script hashes	List<String> // sha256
APK / APK ID	String
Is_hash	Boolean
APK metadata	FastApk

Design: Properties

- Fault tolerance:
  - Broker network error -> retry until tasks published
  - MongoDB network error -> retry until results stored
  - Node crash -> Job still in queue
- Dynamic scheduling
- Add/remove nodes by need

Scripts: Logging




```
{
    "apk meta": {
        "package name": "com.zoner.android.antivirus",
        "version name": "1.9.0",
        "sha256": "88e53942dffe88aa1bbaf374b55d761
        "import date": "2014-12-03T10:15:20.701880
        "path": "/Users/nils/master/it_sec2/worksp
        "tag": "zoner"
    },
    "script meta": {
        "name": "ShowLoggingFuncs",
        "sha256": "3f8a83e9abcaf32de59260df690be8b
        "analysis date": "2014-12-07T19:25:31.5884
        "version": "0.1"
    },
    "demo": {
        "logged": {
            "normal": "some value",
            "bool": true,
            "enum": [
                "list element"
            ]
        },
        "unlogged": {
            "normal": null,
            "bool": false,
            "enum": []
        }
    }
}
```

```
from androlyzelab.model.script.AndroScript import AndroScript

CAT_UNLOGGED = "demo", "unlogged"
CAT_LOGGED = "demo", "logged"

class ShowLoggingFuncs(AndroScript):

  VERSION = "0.1"

  def _analyze(self, apk, dalvik_vm_format, vm_analysis,
    gvm_analysis, *args, **kwargs):
    res = self.res

    # register structure
    res.register_keys(["normal"], *CAT_LOGGED)
    res.register_keys(["normal"], *CAT_UNLOGGED)
    res.register_bool_keys(["bool"], *CAT_LOGGED)
    res.register_bool_keys(["bool"], *CAT_UNLOGGED)
    res.register_enum_keys(["enum"], *CAT_LOGGED)
    res.register_enum_keys(["enum"], *CAT_UNLOGGED)

    # log
    res.log("normal", "some value", *CAT_LOGGED)
    res.log_true("bool", *CAT_LOGGED)
    res.log_append_to_enum("enum", "list element", *CAT_LOGGED)

if __name__ == '__main__':
  for res in AndroScript.test(ShowLoggingFuncs, ["a2dp.Vol.apk"]):
    print res
    print res.write_to_json()
```

Scripts: ClassDetails

ClassDetails.py — UNREGISTERED

```python
from androlyzelab.model.script.AndroScript import AndroScript

# categories
CAT_CLASS_DETAILS = "class details"
CAT_METHODS = "methods"
CAT_FIELDS = "fields"

class ClassDetails(AndroScript):
    ''' Retrieve all classes and their methods and fields '''

    VERSION = "0.1"

    def _analyze(self, apk, dalvik_vm_format, vm_analysis, gvm_analysis, *args, **kwargs):
        res = self.res

        # dvm stuff
        # list<ClassDefItem>
        classes = dalvik_vm_format.get_classes()

        # run over classes
        for c in classes:
            ROOT_CAT = (CAT_CLASS_DETAILS, c.name)
            res.register_keys([CAT_METHODS, CAT_FIELDS], *ROOT_CAT)

            # list<EncodedMethod>
            methods = c.get_methods()
            res.log(CAT_METHODS, [mn.name for mn in methods], *ROOT_CAT)

            # list<EncodedField>
            fields = c.get_fields()
            res.log(CAT_FIELDS, [fn.name for fn in fields], *ROOT_CAT)

    ############################################################
    #---Options
    ############################################################

    def needs_dalvik_vm_format(self):
        return True

if __name__ == '__main__':
    for res in AndroScript.test(ClassDetails, ["../../../../testenv/apks/a2dp.Vol.apk"]):
        print res
        print res.write_to_json()
```

Line 51, Column 34 — Spaces: 4 — Python

Scripts: Options

ScriptTemplate.py — UNREGISTERED

```
31  ##############################################################
32  #---Script requirements
33  ##############################################################
34
35  def needs_dalvik_vm_format(self):
36      return False
37
38  def needs_vmanalysis(self):
39      return False
40
41  def needs_gvmanalysis(self):
42      return False
43
44  def needs_xref(self):
45      return False
46
47  def needs_dref(self):
48      return False
49
50  ##############################################################
51  #---Options
52  ##############################################################
53
54  def create_script_stats(self):
55      return False
56
57  def is_big_res(self):
58      return False
```

Disassemble

Analyze bytecode

CFG ➔ cross refs.

Function refs.

Data refs.

Show timings

Result ≥ 16MB ➔gridFS

Scripts: Chaining

- Do it like in *software engineering*
  - Keep it modular! And chain by need!
- Options:
  - Log errors of chained scripts
  - Skip/continue if one script failed

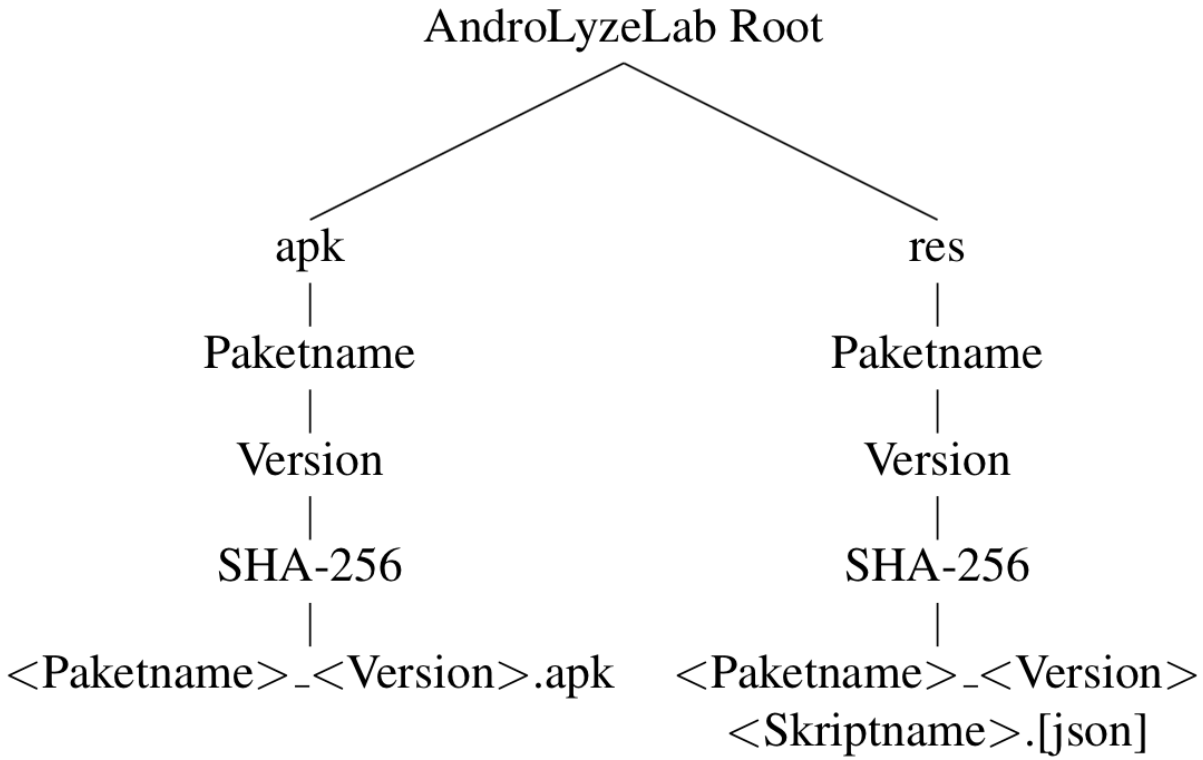
chained_script_meta.json UNREGISTERED

```
...
"ChainedScript": {
    "scripts": [
        "ClassListing",
        "ClassDetails",
        "Disassembly"
    ],
    "successful": [
        "ClassListing",
        "ClassDetails"
    ],
    "failures": [
        {
            "Disassembly": [
                "Traceback (most recent call last):\n",
                ...
                "IndexError: pop from empty list\n"
            ]
        }
    ]
}
```

Scripts: Built-in

1. Manifest
   1. Activities
   2. Services
   3. Files
   4. Intents
   5. Libs
   6. Permissions
   7. ContentProviders
2. Bytecode
   1. ClassDetails
   2. ClassListing
3. Source code
   1. CodePermissions
   2. Decompile(Text)
4. Misc
   1. SSL
   2. GraphAnalysis
   3. GVMAnalysisExample
   4. AnalyzeFrameworks

Storage: Filesystem



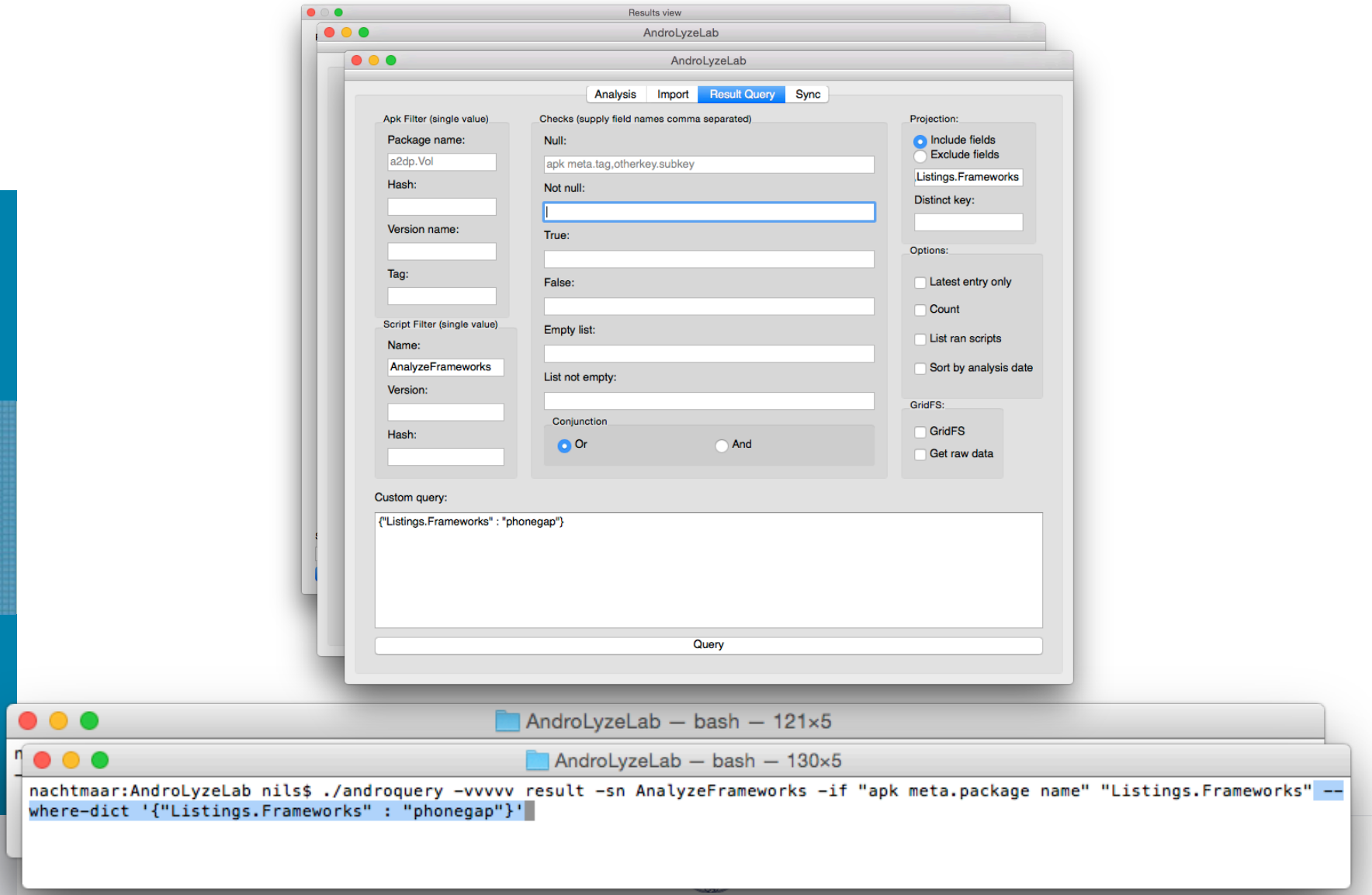
Storage: MongoDB

1. Storage via BSON
2. MongoDB
    1. Authentication: plain vs. credentials vs. credentials + SSL/TLS
    2. Document limitation: 16MB
        1. Use gridfs!
        2. Split files into chunks
        3. Stores files binary with meta data
        4. Can still query metadata
        5. Store big files:
            1. Method call graph
            2. Decompiled code

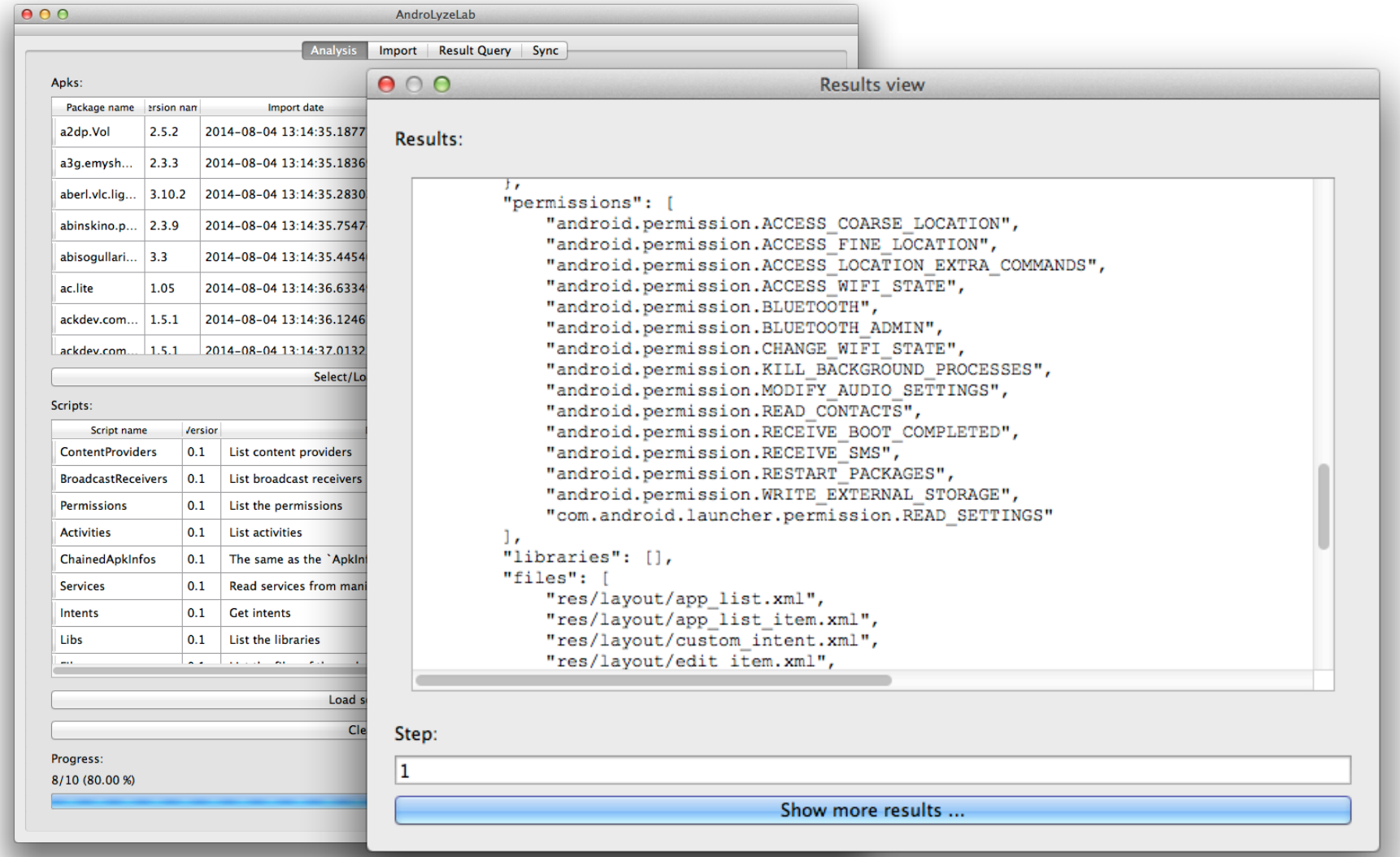
Querying MongoDB : Samples

1. find({'$and': [{u'SSL.url_connection_https': True}, {u'hostname_verifier_allow_all': True}], 'script meta.name': u'SSL'}, {'_id': 0})
2. find({u'Listings.Frameworks': {'$ne': None}, 'script meta.name': u'AnalyzeFrameworks'}, {u'Listings.Frameworks': 1, u'apk meta.package name': 1, '_id': 0})

Querying MongoDB : Assistant



User interface – CLI & GUI



Cluster management

- Management via SSH
- Define nodes in config

- Functions:
  - Install AndroLyzeLab and configure it
  - Deploy scripts
  - Start/stop/restart nodes

Experiments: Physical system

Key	Value
Kernel	3.11.0-23-generic
RAM	4x 8GiB DIMM DDR3 Synchronous 1600 MHz (0,6 ns)
SSD	Toshiba MKNSSDCR240GB
HDD	RAID1: 2x 3TB
CPU	Intel(R) Core(TM) i7-4771 CPU @ 3.50GHz
Cores	4
Cache sizes L1/L2/L3	256KiB, 1MiB, 8MiB
Network	Ethernet Connection I217-LM 1Gbit/s

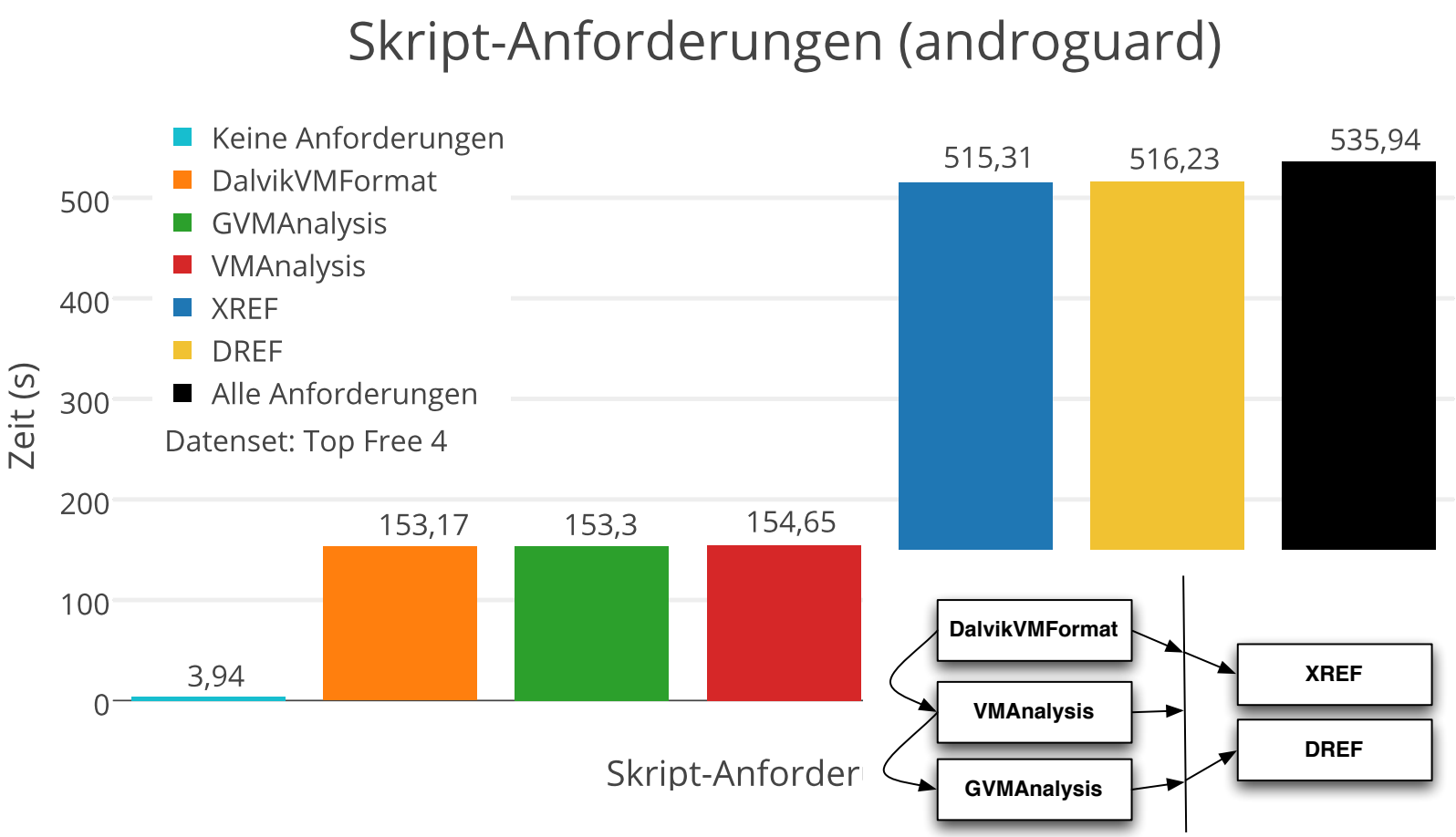
Experiments: APKs

Set	Description	Count	Size (MB)
ApkSet1	Top Free 4	102	1.159
ApkSet2	Top Free 100	2.519	22.315
ApkSet3	Top Free 500	12.689	91.764

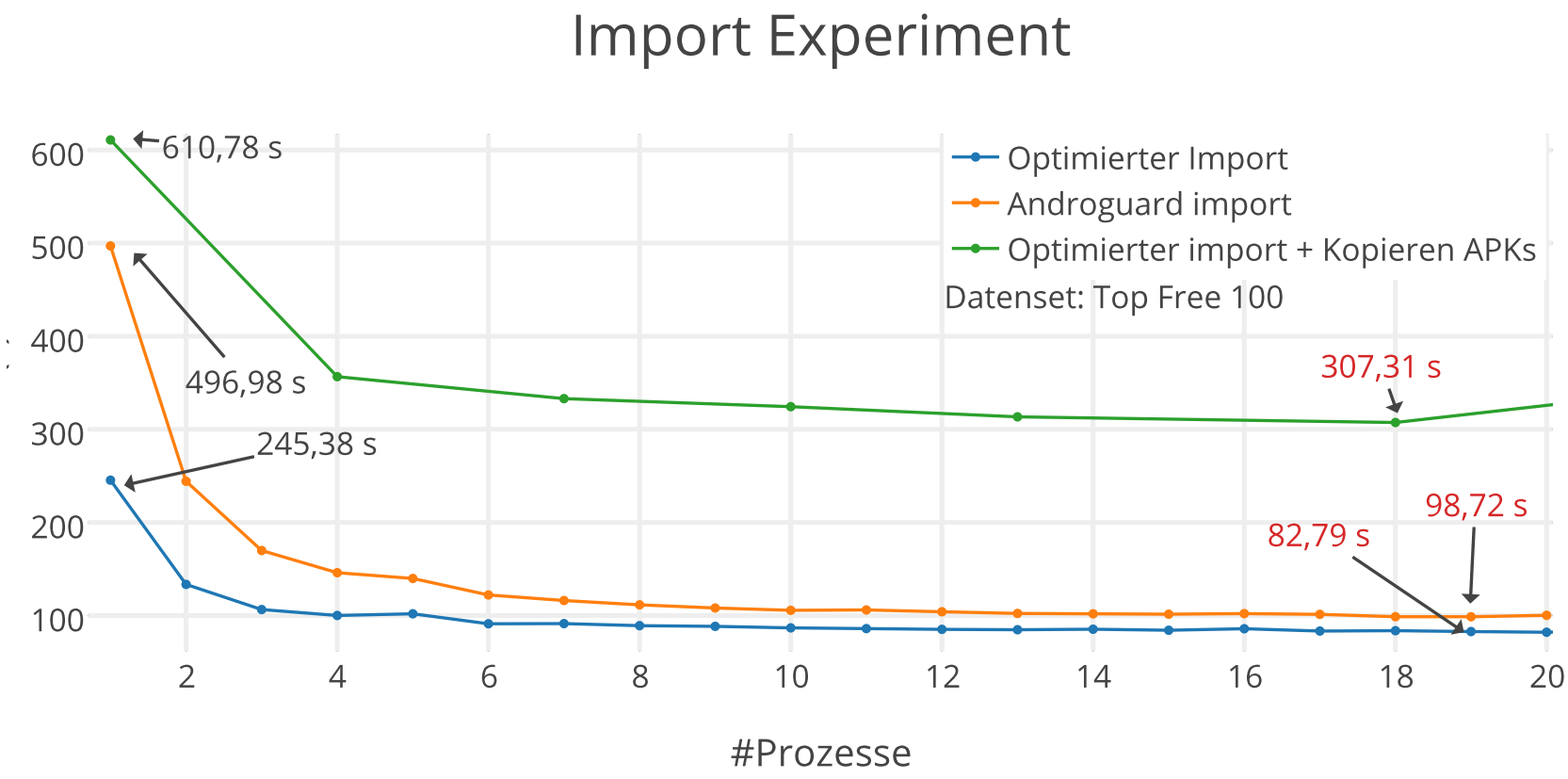
Experiments: Scripts

Set	Requirements	Scripts
Manifest	-	ChainedApkInfos, Files, Libs, Activities, Intents, ContentProviders, Services, BroadcastReceivers, Permissions
Manifest + SSL	XREF	Manifest ∪ {SSL}
Misc1	XREF	Manifest + SSL ∪ {ClassListing, ClassDetails, AnalyzeFrameworks, GVMAnalysisExample}
Misc2	XREF	Misc1 ∪ {Decompile}

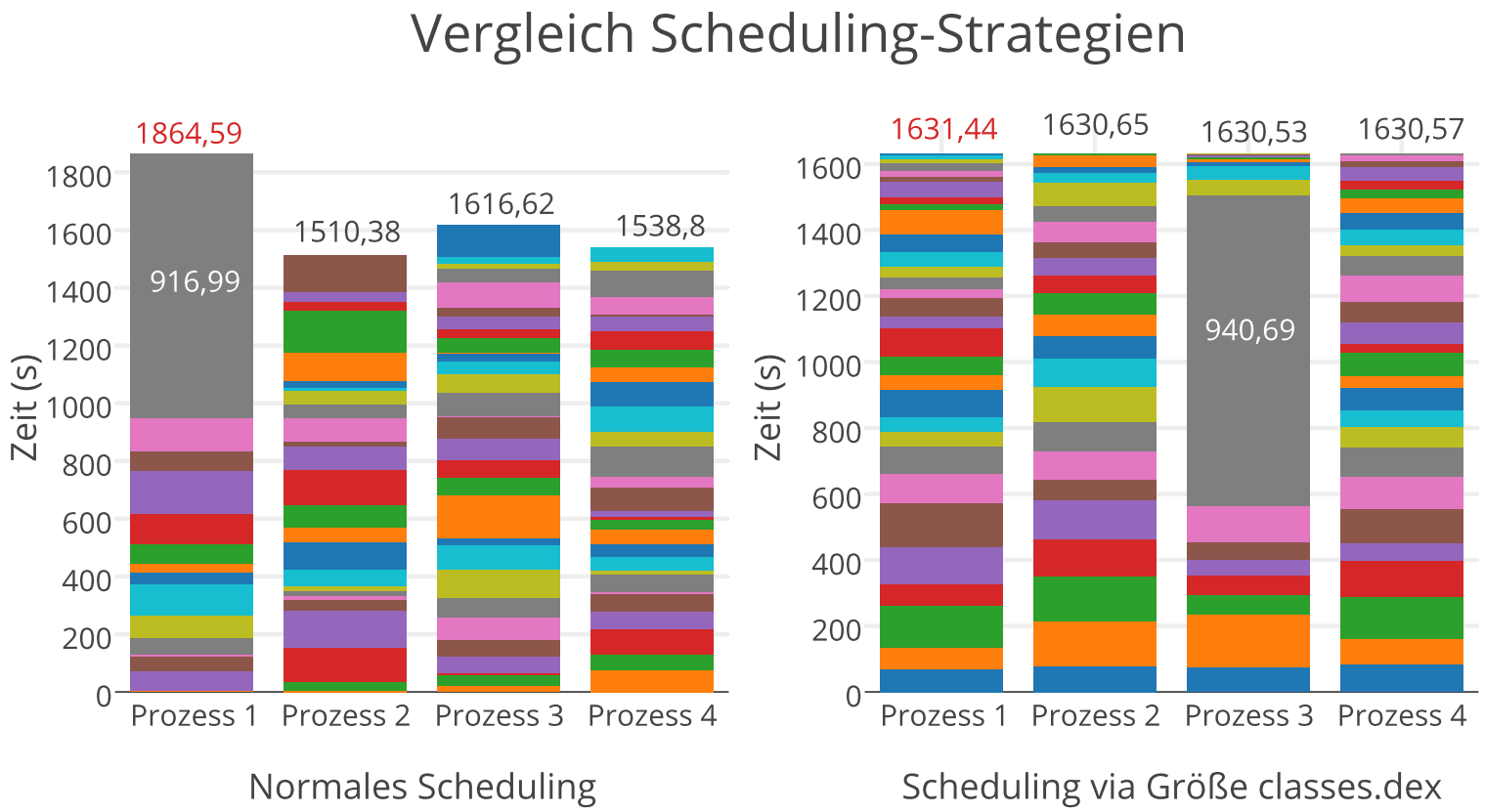
Experiments: #1



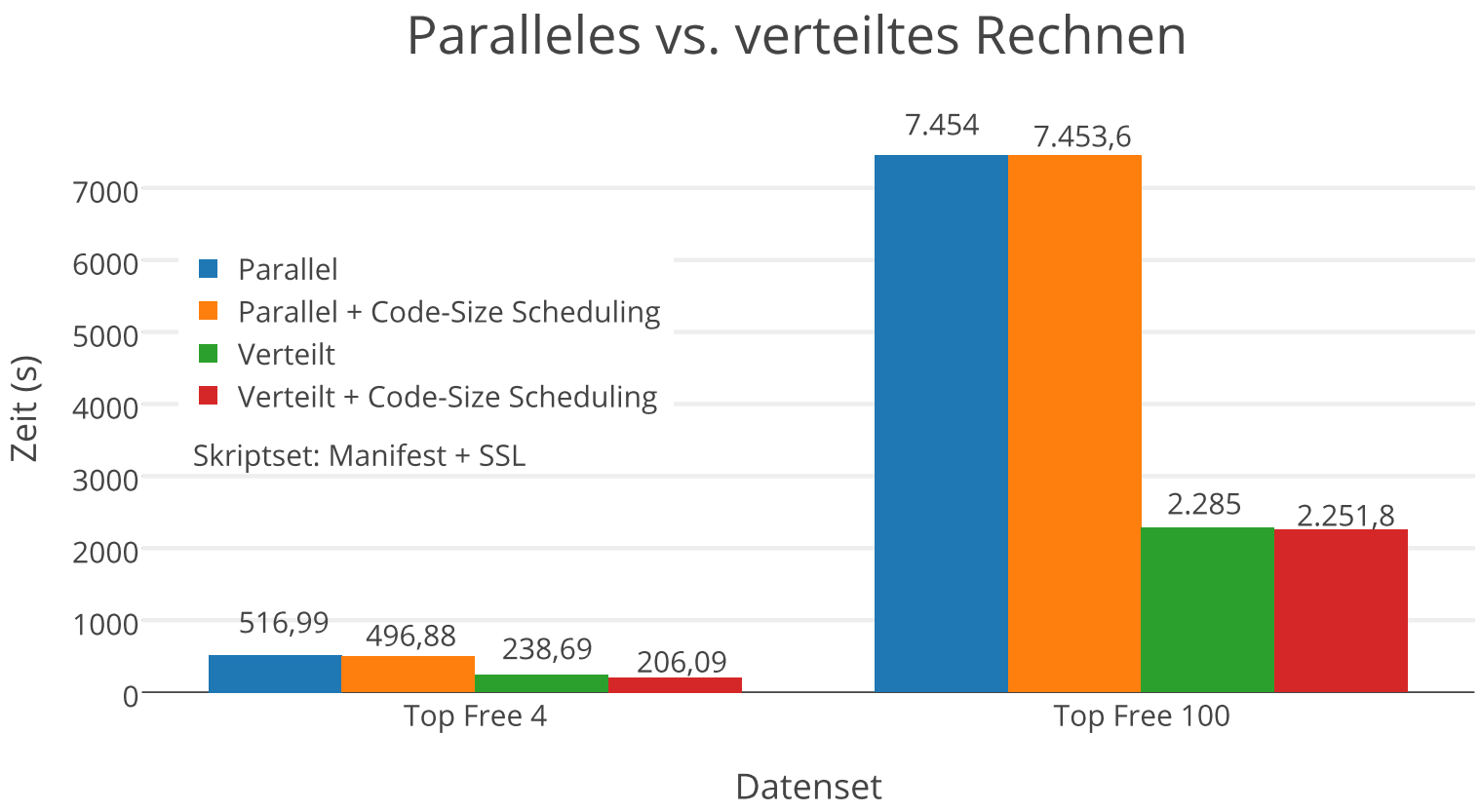
Experiments: #2



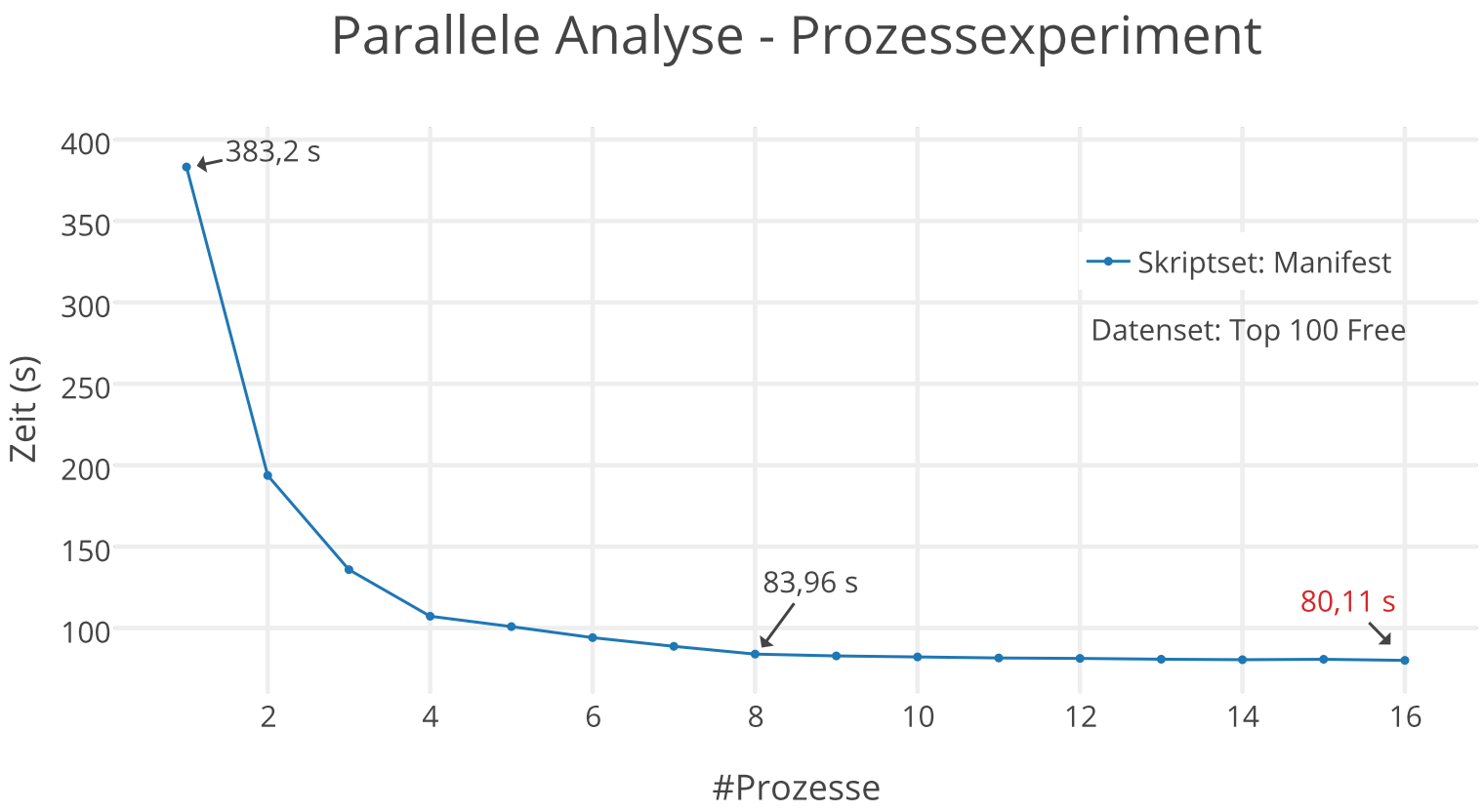
Experiments: #3



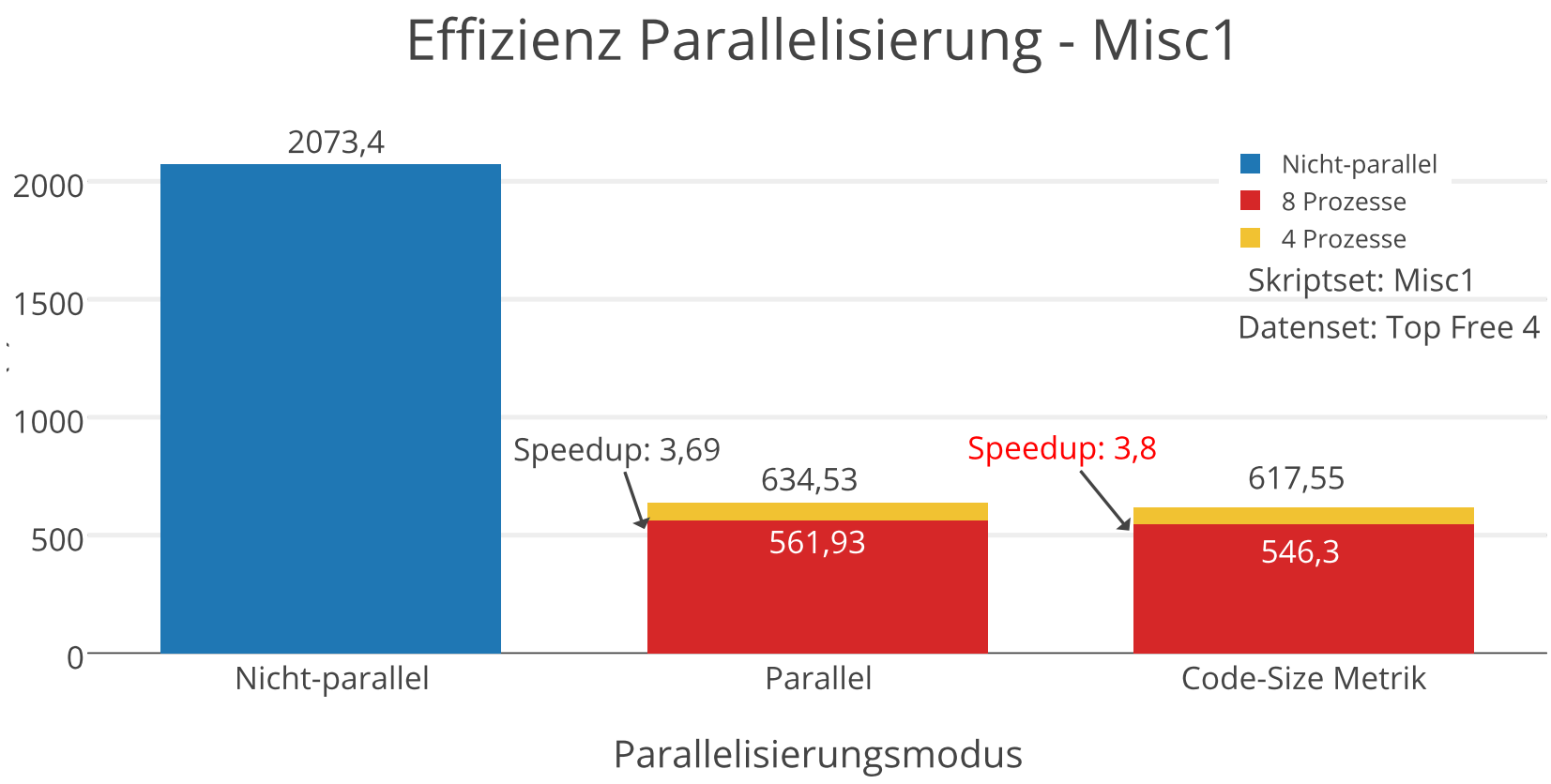
Experiments: #4



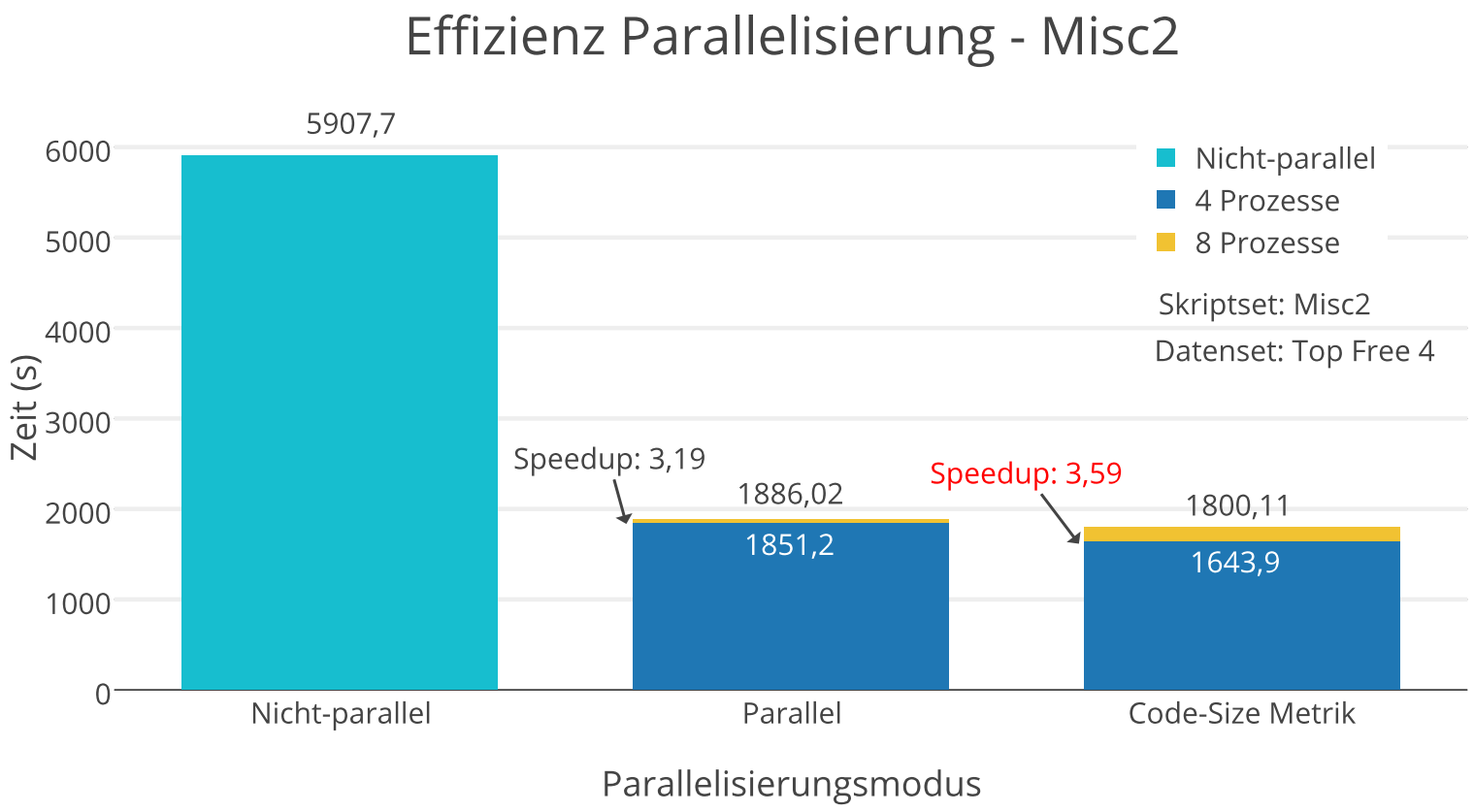
Experiments: #5



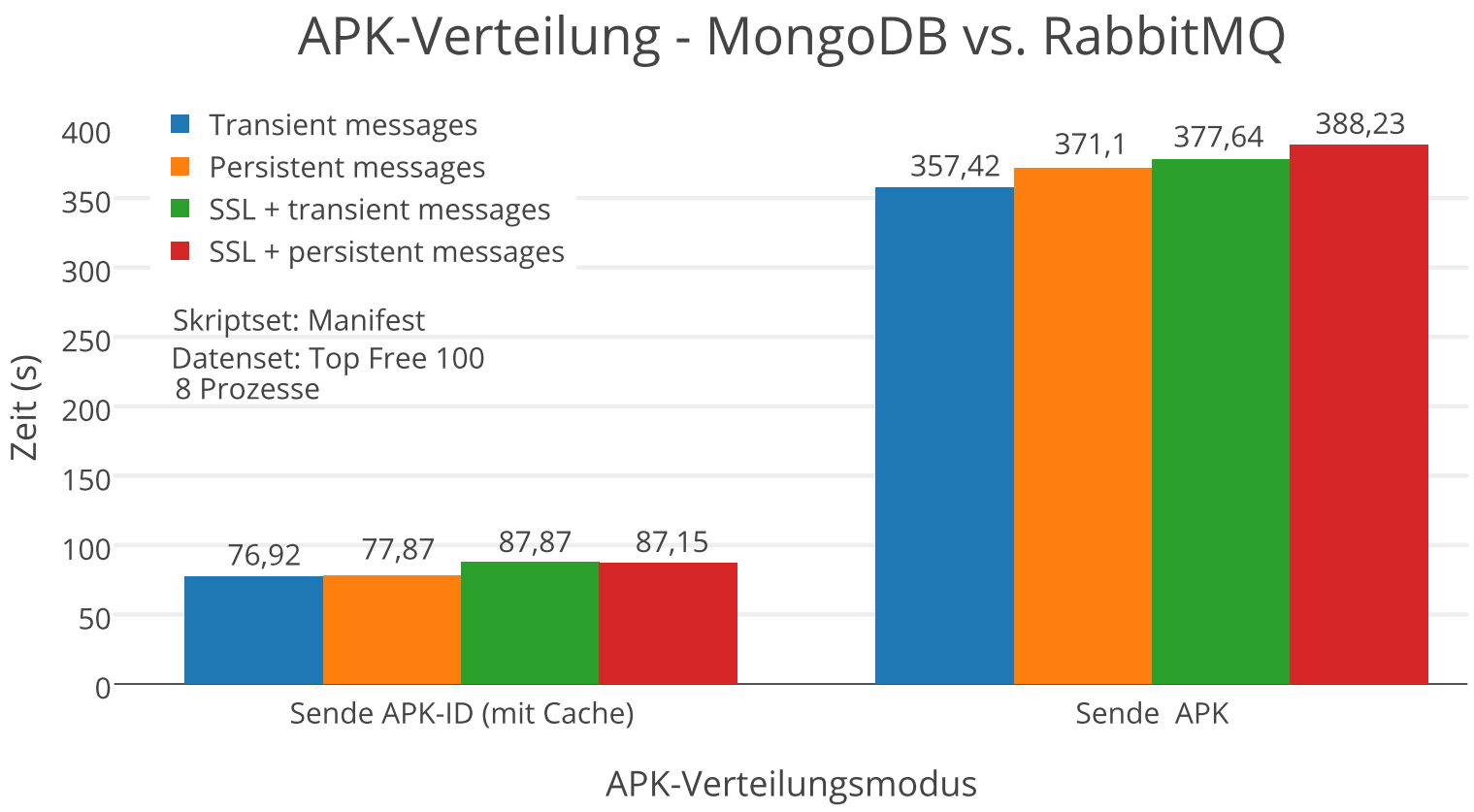
Experiments: #6



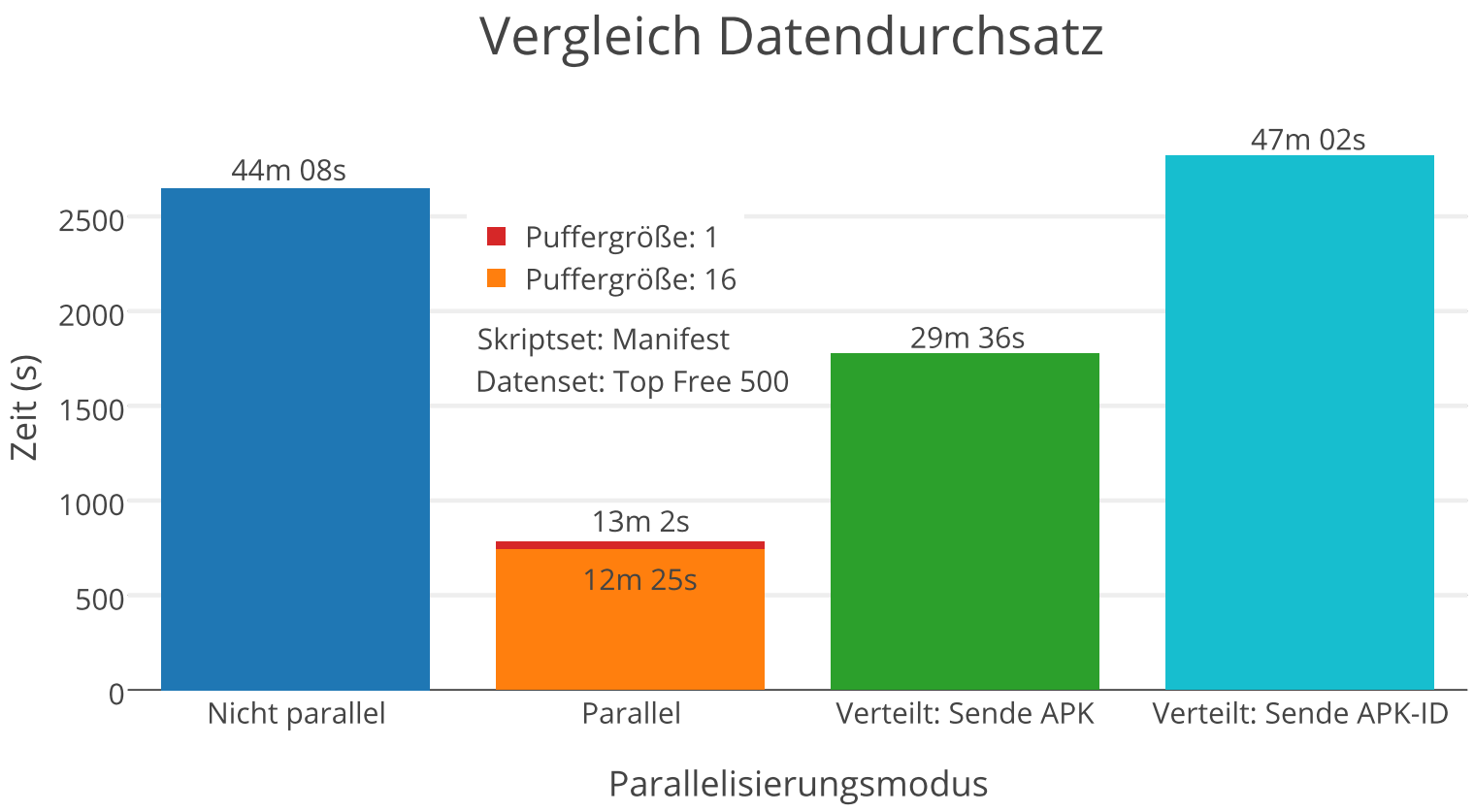
Experiments: #7



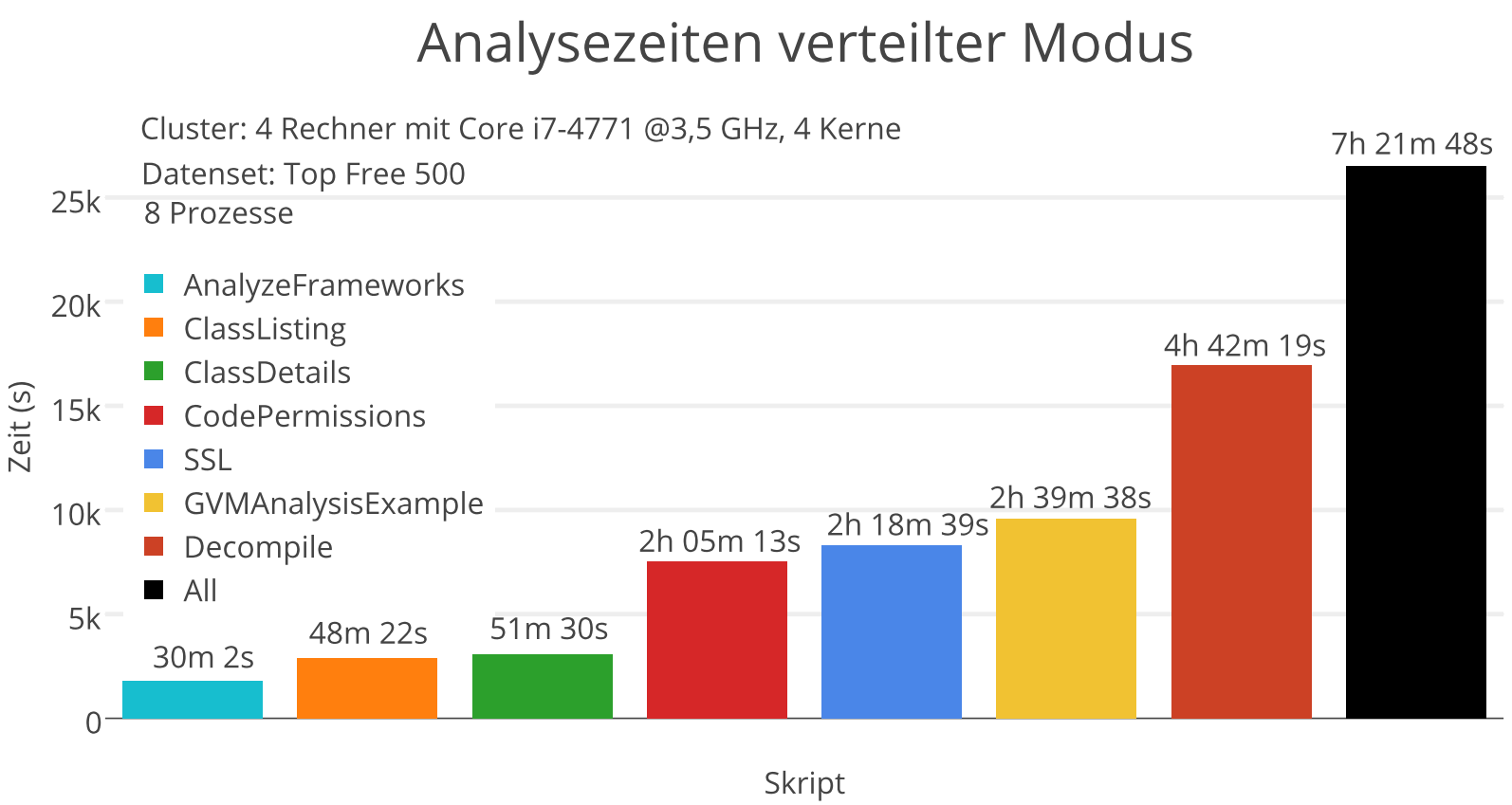
Experiments: #8



Experiments: #9



Experiments: #10



Future work

- Analysis
  - Integrate all androguard based projects
  - AndroWarn
  - CryptoLint
  - MalloDroid
  - Decompiler cache
  - More decompilers
  - Scripts, scripts, scripts:
    - Androguard signatures for malware detection
    - Detect similarities between apps
    - Easier script-requirements
    - Add dynamic analysis functionality
- App statistics

Future work

- Import database:
  - Regular expressions for APK filtering
- Celery:
  - More queues based on node performance
  - Routing via code size and script requirements
  - Implement remote controls -> monitoring
    - Use code-size metric for scheduling as standard
    - Use SSD for APK storage (HDD bottlenecks)

Future work

- Event-based analysis
  - Register event
  - If action triggers on next scheduled analysis
  - Inform via E-Mail
  - Example: Check if ZonerAV still vulnerable to SSL-MITM
- Message sending
  - Parallelization of message serialization
  - Integrate scripts into message ➔ no need for script deployment anymore
- MongoDB
  - Can we further improve performance?
  - In-memory database
  - Use sharding