quark-engine

Release v21.8.1

Oct 08, 2022

Contents:

Quark Engine is an *open source* software for automating analysis of suspicious Android application. To do so it makes use of custom Dalvik Bytecode Loader and unique scoring system that detect malicious behaviors and calculate threat level within seconds.

This guide will explain how to set up Quark, use it, and customize it.

CHAPTER 1

Installation

This chapter explains how to write Quark's code and how to contribute.

1.1 Installing Quark-Engine

PyPi:

```
$ pip3 install -U quark-engine
```

Install from Source:

```
$ git clone https://github.com/quark-engine/quark-engine.git
$ cd quark-engine/
$ pipenv install --skip-lock
$ pipenv shell
```

Run the help cmd of quark:

```
$ quark --help
```

Once you see the following msg, then you're all set:

```
Usage: quark [OPTIONS]

  Quark is an Obfuscation-Neglect Android Malware Scoring System

Options:
  -s, --summary TEXT              Show summary report. Optionally specify the
                                  name of a rule/label
  -d, --detail TEXT               Show detail report. Optionally specify the
                                  name of a rule/label
  -o, --output FILE               Output report in JSON
  -a, --apk FILE                  APK file  [required]
```

(continues on next page)

(continued from previous page)

```
  -r, --rule PATH                 Rules directory  [default:
                                  /Users/$USER/.quark-engine/quark-rules]
  -g, --graph                     Create call graph to call_graph_image
                                  directory
  -c, --classification            Show rules classification
  -t, --threshold [100|80|60|40|20]
                                  Set the lower limit of the confidence
                                  threshold
  -i, --list [all|native|custom]  List classes, methods and descriptors
  -p, --permission                List Android permissions
  -l, --label [max|detailed]      Show report based on label of rules
  -C, --comparison                Behaviors comparison based on max confidence
                                  of rule labels
  --core-library [androguard|rizin]
                                  Specify the core library used to analyze an
                                  APK
  --multi-process INTEGER RANGE   Allow analyzing APK with N processes,
                                  where N doesn't exceeds the number of usable CPUs -␣
↪1
                                  to avoid memory exhaustion.
  --version                       Show the version and exit.
  --help                          Show this message and exit.
```

To learn how to scan multiple samples in a directory, please have a look at *Directory Scanning*

1.2 Testing Quark-Engine

Firstly let's fetch some apk examples.

```
$ git clone https://github.com/quark-engine/apk-malware-samples.git
```

Then we could test one of the apk in *apk-malware-samples* by the rules *quark-rules*. For example:

```
$ quark -a Ahmyth.apk -s
```

1.2.1 Running in Docker

We could build the corresponding docker image by this command:

```
docker build . -t quark
```

When the image is ready, let's run an example to test the image:

You may also interactively use quark in the docker container. For example:

```
$ docker run -v $(pwd):/tmp -it quark bash
(in-docker): /app/quark# cd /tmp
(in-docker)::/tmp# quark -a Ahmyth.apk -s
```

CHAPTER 2

Quark Script

2.1 Ecosystem for Mobile Security Tools

2.1.1 Innovative & Interactive

The goal of Quark Script aims to provide an innovative way for mobile security researchers to analyze or **pentest** the targets.

Based on Quark, we integrate decent tools as Quark Script APIs and make them exchange valuable intelligence to each other. This enables security researchers to **interact** with staged results and perform **creative** analysis with Quark Script.

2.1.2 Dynamic & Static Analysis

In Quark script, we integrate not only static analysis tools (e.g. Quark itself) but also dynamic analysis tools (e.g. objection).

2.1.3 Re-Usable & Sharable

Once the user creates a Quark script for specific analysis scenario. The script can be used in another targets. Also, the script can be shared to other security researchers. This enables the exchange of knowledges.

2.1.4 More APIs to come

Quark Script is now in a beta version. We'll keep releasing practical APIs and analysis scenarios.

2.2 Introduce of Quark Script APIs

2.2.1 Rule(rule.json)

- **Description**: Making detection rule a rule instance
- **params**: Path of a single Quark rule
- **return**: Quark rule instance

2.2.2 runQuarkAnalysis(SAMPLE_PATH, ruleInstance)

- **Description**: Given detection rule and target sample, this instance runs the basic Quark analysis.
- **params:**
    1. SAMPLE_PATH: Target file
    2. ruleInstance: Quark rule object
- **return**: quarkResult instance

2.2.3 quarkResultInstance.behaviorOccurList

- **Description**: List that stores instances of detected behavior in different part of the target file.
- **params**: none
- **return**: detected behavior instance

2.2.4 quarkResultInstance.getAllStrings(none)

- **Description**: Get all strings inside the target APK file.
- **params**: none
- **return**: python list containing all strings inside the target APK file.

2.2.5 quarkResultInstance.findMethodInCaller(callerMethod, targetMethod)

- **Description**: Check if target method is in caller method.
- **params:**
    1. callerMethod: python list contains class name, method name and descriptor of caller method.
    2. targetMethod: python list contains class name, method name and descriptor of target method.
- **return**: True/False

2.2.6 behaviorInstance.firstAPI.fullName

- **Description**: Show the name of the first key API called in this behavior.
- **params**: none
- **return**: API name

2.2.7 behaviorInstance.secondAPI.fullName

- **Description**: Show the name of the second key API called in this behavior.
- **params**: none
- **return**: API name

2.2.8 behaviorInstance.hasUrl(none)

- **Description**: Check if the behavior contains urls.
- **params**: none
- **return**: python list containing all detected urls.

2.2.9 behaviorInstance.methodCaller

- **Description**: Find method who calls this behavior (API1 & API2).
- **params**: none
- **return**: method instance

2.2.10 behaviorInstance.getParamValues(none)

- **Description**: Get parameter values that API1 sends to API2 in the behavior.
- **params**: none
- **return**: python list containing parameter values.

2.2.11 behaviorInstance.isArgFromMethod(targetMethod)

- **Description**: Check if there are any arguments from the target method.
- **params:**
    1. targetMethod: python list contains class name, method name, and descriptor of target method
- **return**: True/False

2.2.12 methodInstance.getXrefFrom(none)

- **Description**: Find out who call this method.
- **params**: none
- **return**: python list containing caller methods.

2.2.13 methodInstance.getXrefTo(none)

- **Description**: Find out who this method called.
- **params**: none
- **return**: python list containing tuples (callee methods, index).

2.2.14 methodInstance.getArguments(none)

- **Description**: Get arguments from method.
- **params**: none
- **return**: python list containing arguments.

2.2.15 Objection(host)

- **Description**: Create an instance for Objection (dynamic analysis tool).
- **params**: Monitoring IP:port
- **return**: objection instance

2.2.16 objInstance.hookMethod(method, watchArgs, watchBacktrace, watchRet)

- **Description**: Hook the target method with Objection.
- **params:**
    1. method: the tagrget API. (type: str or method instance)
    2. watchArgs: Return Args information if True. (type: boolean)
    3. watchBacktrace: Return backtrace information if True. (type: boolean)
    4. watchRet: Return the return information of the target API if True. (type: boolean)
- **return**: none

2.2.17 runFridaHook(apkPackageName, targetMethod, methodParamTypes, secondToWait)

- **Description**: Track calls to the specified method for given seconds.
- **params:**
    1. apkPackageName: the package name of the target APP
    2. targetMethod: the target API
    3. methodParamTypes: string that holds the parameters used by the target API
    4. secondToWait: seconds to wait for method calls, defaults to 10
- **return**: FridaResult instance

2.2.18 checkClearText(inputString)

- **Description**: Check the decrypted value of the input string.
- **params:**
    1. inputString: string to be checked
- **return**: the decrypted value

2.2.19 getActivities(samplePath)

- **Description**: Get activities from the manifest of target sample.
- **params:**
    1. samplePath: the file path of target sample
- **return**: python list containing activities

2.2.20 activityInstance.hasIntentFilter(none)

- **Description**: Check if the activity has an intent-filter.
- **params**: none
- **return**: True/False

2.2.21 activityInstance.isExported(none)

- **Description**: Check if the activity set `android:exported=true`.
- **params**: none
- **return**: True/False

2.3 Analyzing real case (InstaStealer) using Quark Script

2.3.1 Quark Script that dynamic hooks the method containing urls

The scenario is simple! We'd like to dynamic hooking the methods in the malware that contains urls. We can use APIs above to write Quark Script.

```
from quark.script import runQuarkAnalysis, Rule
from quark.script.objection import Objection

SAMPLE_PATH = "6f032.apk"
RULE_PATH = "00211.json"

ruleInstance = Rule(RULE_PATH)
quarkResult = runQuarkAnalysis(SAMPLE_PATH, ruleInstance)

for behaviorInstance in quarkResult.behaviorOccurList:
    detectedUrl = behaviorInstance.hasUrl()
```

(continues on next page)

(continued from previous page)

```
    if detectedUrl:
        print(f"\nDetected Behavior -> {ruleInstance.crime}")
        print(f"\nDetected Url -> {detectedUrl}")

        method = behaviorInstance.methodCaller
        print(f"\nThe detected behavior was called by -> {method.fullName}")

        print("\nAttempt to hook the method:")
        obj = Objection("127.0.0.1:8888")

        obj.hookMethod(method,
                       watchArgs=True,
                       watchBacktrace=True,
                       watchRet=True)
        print(f"\tHook -> {method.fullName}")

        for methodCaller in method.getXrefFrom():
            obj.hookMethod(methodCaller,
                           watchArgs=True,
                           watchBacktrace=True,
                           watchRet=True)
            print(f"\tHook -> {methodCaller.fullName}")

        for methodCallee, _ in method.getXrefTo():
            obj.hookMethod(methodCallee,
                           watchArgs=True,
                           watchBacktrace=True,
                           watchRet=True)
            print(f"\tHook -> {methodCallee.fullName}")

print("\nSee the hook results in Objection's terminal.")
```

Note: Please make sure you have the dynamic analysis environment ready before executing the script.

1. Objection installed and running. Check the guideline here.
2. Android Virtual Machine with frida installed. Check the guideline here.
3. Or a rooted Android Device (Google Pixel 6) with frida installed. Check the root guideline here, frida install guideline is the same with Android Virtual Machine.

2.3.2 Quark Script Result

```
Detected Behavior -> Open an URL in Wevbiew

Detected Url -> ['https://insfreefollower.com/']

The detected behavior was called by -> Lio/kodular/djmunsheer/Instagram_71/Screen2; Screen2$Initialize ()Ljava/lang/Object;

Attempt to hook the method:
        Hook -> Lio/kodular/djmunsheer/Instagram_71/Screen2; Screen2$Initialize ()Ljava/lang/Object;
        Hook -> Lio/kodular/djmunsheer/Instagram_71/Screen2$frame; apply0 (Lgnu/expr/ModuleMethod;)Ljava/lang/Object;
        Hook -> Lcom/google/youngandroid/runtime; setThisForm ()V
        Hook -> Lgnu/lists/LList; list1 (Ljava/lang/Object;)Lgnu/lists/Pair;
        Hook -> Lcom/google/youngandroid/runtime; callComponentMethod (Ljava/lang/Object; Ljava/lang/Object; Ljava/lang/Object; Ljava/lang/Object;)Ljava/lang/Object;

See the hook results in Objection's terminal.
```

2.3.3 Logs on the Objection terminal (hooking)

```
(agent) Attempting to watch class io.kodular.djmunsheer.Instagram_71.Screen2 and method Screen2$Initialize.
(agent) Will filter for method overload with arguments:
(agent) Hooking io.kodular.djmunsheer.Instagram_71.Screen2.Screen2$Initialize()
(agent) Registering job 312773. Type: watch-method for: io.kodular.djmunsheer.Instagram_71.Screen2.Screen2$Initialize
(agent) Attempting to watch class io.kodular.djmunsheer.Instagram_71.Screen2$frame and method apply0.
(agent) Will filter for method overload with arguments: gnu.expr.ModuleMethod
(agent) Hooking io.kodular.djmunsheer.Instagram_71.Screen2$frame.apply0(gnu.expr.ModuleMethod)
(agent) Registering job 656187. Type: watch-method for: io.kodular.djmunsheer.Instagram_71.Screen2$frame.apply0
(agent) Attempting to watch class gnu.lists.LList and method list1.
(agent) Will filter for method overload with arguments: java.lang.Object
(agent) Hooking gnu.lists.LList.list1(java.lang.Object)
(agent) Registering job 584100. Type: watch-method for: gnu.lists.LList.list1
(agent) Attempting to watch class com.google.youngandroid.runtime and method setThisForm.
(agent) Will filter for method overload with arguments:
(agent) Hooking com.google.youngandroid.runtime.setThisForm()
(agent) Registering job 811719. Type: watch-method for: com.google.youngandroid.runtime.setThisForm
(agent) Attempting to watch class com.google.youngandroid.runtime and method callComponentMethod.
(agent) Will filter for method overload with arguments: java.lang.Object,java.lang.Object,java.lang.Object,java.lang.Object
(agent) Hooking com.google.youngandroid.runtime.callComponentMethod(java.lang.Object, java.lang.Object, java.lang.Object, java.lang.Object)
(agent) Registering job 942893. Type: watch-method for: com.google.youngandroid.runtime.callComponentMethod
```

2.3.4 Method (callComponentMethod) with urls is detected triggered!

```
(agent) [483202] Called com.google.youngandroid.runtime.callComponentMethod(java.lang.Object, java.lang.Object, java.lang.Object, java.lang.Object)
(agent) [483202] Backtrace:
        com.google.youngandroid.runtime.callComponentMethod(Native Method)
        io.kodular.djmunsheer.Instagram_71.Screen2.Screen2$Initialize(Screen2.yail:22)
        io.kodular.djmunsheer.Instagram_71.Screen2.Screen2$Initialize(Native Method)
        io.kodular.djmunsheer.Instagram_71.Screen2$frame.apply0(Screen2.yail:523)
        io.kodular.djmunsheer.Instagram_71.Screen2$frame.apply0(Native Method)
        gnu.expr.ModuleBody.applyN(ModuleBody.java:226)
        gnu.expr.ModuleMethod.applyN(ModuleMethod.java:216)
        gnu.kawa.functions.ApplyToArgs.applyN(ApplyToArgs.java:139)
        gnu.kawa.functions.Apply.applyN(Apply.java:70)
        gnu.mapping.ProcedureN.apply2(ProcedureN.java:39)
        io.kodular.djmunsheer.Instagram_71.Screen2.dispatchEvent(Screen2.yail:10125)
        com.google.appinventor.components.runtime.EventDispatcher.hxYOFxFjLpN1maJuWNxUV40nExCGxsxkDPOTgtzMu4zlZCQb3bPlKsXo1SYJg6ME(SourceFile:243)
        com.google.appinventor.components.runtime.EventDispatcher.dispatchEvent(SourceFile:199)
        com.google.appinventor.components.runtime.Form$16.run(SourceFile:1046)
        android.os.Handler.handleCallback(Handler.java:938)
        android.os.Handler.dispatchMessage(Handler.java:99)
        android.os.Looper.loopOnce(Looper.java:201)
        android.os.Looper.loop(Looper.java:288)
        android.app.ActivityThread.main(ActivityThread.java:7839)
        java.lang.reflect.Method.invoke(Native Method)
        com.android.internal.os.RuntimeInit$MethodAndArgsCaller.run(RuntimeInit.java:548)
        com.android.internal.os.ZygoteInit.main(ZygoteInit.java:1003)

(agent) [483202] Arguments com.google.youngandroid.runtime.callComponentMethod(Web_Viewer1, GoToUrl, (https://insfreefollower.com/), (text))
(agent) [483202] Return Value: (none)
```

2.4 Detect CWE-798 in Android Application (ovaa.apk)

This scenario seeks to find hard-coded credentials in the APK file. See CWE-798 for more details.

Let's use this APK and the above APIs to show how Quark script find this vulnerability.

First, we design a detection rule `findSecretKeySpec.json` to spot on behavior uses method SecretKeySpec. Then, we get all the parameter values that input to this method. From the returned parameter values, we identify it's a AES key and parse the key out of the values. Finally, we dump all strings in the APK file and check if the AES key is in the strings. If the answer is YES, BINGO!!! We find hard-coded credentials in the APK file.

2.4.1 Quark Scipt: CWE-798.py

```
import re
from quark.script import runQuarkAnalysis, Rule
```

(continues on next page)

(continued from previous page)

```
SAMPLE_PATH = "ovaa.apk"
RULE_PATH = "findSecretKeySpec.json"

ruleInstance = Rule(RULE_PATH)
quarkResult = runQuarkAnalysis(SAMPLE_PATH, ruleInstance)

for secretKeySpec in quarkResult.behaviorOccurList:

    allStrings = quarkResult.getAllStrings()

    firstParam = secretKeySpec.getParamValues()[0]
    secondParam = secretKeySpec.getParamValues()[1]

    if secondParam == "AES":
        AESKey = re.findall(r'\((.*?)\)', firstParam)[1]

    if AESKey in allStrings:
        print(f"Found hard-coded {secondParam} key {AESKey}")
```

2.4.2 Quark Rule: findSecretKeySpec.json

```
{
    "crime": "Detect APK using SecretKeySpec.",
    "permission": [],
    "api": [
        {
            "descriptor": "()[B",
            "class": "Ljava/lang/String;",
            "method": "getBytes"
        },
        {
            "descriptor": "([BLjava/lang/String;)V",
            "class": "Ljavax/crypto/spec/SecretKeySpec;",
            "method": "<init>"
        }
    ],
    "score": 1,
    "label": []
}
```

2.4.3 Quark Script Result

```
$ python3 findSecretKeySpec.py

Found hard-coded AES key 49u5gh249gh24985ghf429gh4ch8f23f
```

2.4.4 Hard-Coded AES key in the APK file

```
const-string v2, "49u5gh249gh24985ghf429gh4ch8f23f"
```

(continues on next page)

(continued from previous page)

```
invoke-virtual {v2}, Ljava/lang/String;->getBytes()[B

move-result-object v2

invoke-direct {v1, v2, v0}, Ljavax/crypto/spec/SecretKeySpec;-><init>([BLjava/lang/
↪String;)V
```

2.5 Detect CWE-94 in Android Application (ovaa.apk)

This scenario seeks to find code injection in the APK file. See CWE-94 for more details.

Let's use this APK and the above APIs to show how Quark script find this vulnerability.

First, we design a detection rule `loadExternalCode.json` to spot on behavior uses method createPackageContext. Then, we find the caller method who calls the createPackageContext. Finally, we check if method checkSignatures is called in the caller method for verification.

2.5.1 Quark Scipt: CWE-94.py

```
from quark.script import runQuarkAnalysis, Rule

SAMPLE_PATH = "ovaa.apk"
RULE_PATH = "loadExternalCode.json"

targetMethod = [
        "Landroid/content/pm/PackageManager;",
        "checkSignatures",
        "(Ljava/lang/String;Ljava/lang/String;)I"
        ]

ruleInstance = Rule(RULE_PATH)
quarkResult = runQuarkAnalysis(SAMPLE_PATH, ruleInstance)

for ldExternalCode in quarkResult.behaviorOccurList:

    callerMethod = [
            ldExternalCode.methodCaller.className,
            ldExternalCode.methodCaller.methodName,
            ldExternalCode.methodCaller.descriptor
            ]

    if not quarkResult.findMethodInCaller(callerMethod, targetMethod):
        print(f"\nMethod: {targetMethod[1]} not found!")
        print(f"CWE-94 is detected in {SAMPLE_PATH}")
```

2.5.2 Quark Rule: loadExternalCode.json

```
{
    "crime": "Load external code from other APK.",
    "permission": [],
    "api": [
```

(continues on next page)

(continued from previous page)

```
        {
            "descriptor": "(Ljava/lang/String;I)Landroid/content/Context;",
            "class": "",
            "method": "createPackageContext"
        },
        {
            "descriptor": "(Ljava/lang/String;)Ljava/lang/Class;",
            "class": "Ljava/lang/ClassLoader;",
            "method": "loadClass"
        }
    ],
    "score": 1,
    "label": []
}
```

2.5.3 Quark Script Result

```
$ python3 CWE-94.py

Method: checkSignatures not found!
CWE-94 is detected in ovaa.apk
```

2.6 Detect CWE-921 in Android Application (ovaa.apk)

This scenario seeks to find unsecure storage mechanism of data in the APK file. See CWE-921 for more details.

Let's use this APK and the above APIs to show how Quark script find this vulnerability.

First, we design a detection rule `checkFileExistence.json` to spot on behavior that checks if a file exist on given storage mechanism. Then, we use API `getParamValues()` to get the file path. Finally, CWE-921 is found if the file path contains keyword `sdcard`.

2.6.1 Quark Script CWE-921.py

```
from quark.script import runQuarkAnalysis, Rule

SAMPLE_PATH = "ovaa.apk"
RULE_PATH = "checkFileExistence.json"

ruleInstance = Rule(RULE_PATH)
quarkResult = runQuarkAnalysis(SAMPLE_PATH, ruleInstance)

for existingFile in quarkResult.behaviorOccurList:
    filePath = existingFile.getParamValues()[0]
    if "sdcard" in filePath:
        print(f"This file is stored inside the SDcard\n")
        print(f"CWE-921 is detected in {SAMPLE_PATH}.")
```

2.6.2 Quark Rule: checkFileExistence.json

```
{
    "crime": "Check file existence",
    "permission": [],
    "api": [
        {
            "descriptor": "(Ljava/lang/String;)V",
            "class": "Ljava/io/File;",
            "method": "<init>"
        },
        {
            "descriptor": "()Z",
            "class": "Ljava/io/File;",
            "method": "exists"
        }
    ],
    "score": 1,
    "label": []
}
```

2.6.3 Quark Script Result

```
$ python3 CWE-921.py
This file is stored inside the SDcard

CWE-921 is detected in ovaa.apk.
```

2.7 Detect CWE-312 in Android Application (ovaa.apk)

This scenario seeks to find cleartext storage of sensitive data in the APK file. See CWE-312 for more details.

Let's use this APK and the above APIs to show how Quark script find this vulnerability.

First, we designed a Frida script `agent.js` to hook the target method and get the arguments when the target method is called. Then we hook the method `putString` to catch its arguments. Finally, we use Ciphey to check if the arguments are encrypted.

2.7.1 Quark Script CWE-312.py

```
from quark.script.frida import runFridaHook
from quark.script.ciphey import checkClearText

APP_PACKAGE_NAME = "oversecured.ovaa"

TARGET_METHOD = "android.app." \
                "SharedPreferencesImpl$EditorImpl." \
                "putString"

METHOD_PARAM_TYPE = "java.lang.String," \
                    "java.lang.String"
```

(continues on next page)

(continued from previous page)

```
fridaResult = runFridaHook(APP_PACKAGE_NAME,
                           TARGET_METHOD,
                           METHOD_PARAM_TYPE,
                       secondToWait = 10)

for putString in fridaResult.behaviorOccurList:

    firstParam, secondParam = putString.getParamValues()

    if firstParam in ["email", "password"] and \
        secondParam == checkClearText(secondParam):

        print(f'The CWE-312 vulnerability is found. The cleartext is "{secondParam}"')
```

2.7.2 Frida Script: agent.js

```
// -*- coding: utf-8 -*-
// This file is part of Quark-Engine - https://github.com/quark-engine/quark-engine
// See the file 'LICENSE' for copying permission.

/*global Java, send, rpc*/
function replaceMethodImplementation(targetMethod, classAndMethodName,␣
↪methodParamTypes, returnType) {
    targetMethod.implementation = function () {
        let callEvent = {
            "type": "CallCaptured",
            "identifier": [classAndMethodName, methodParamTypes, returnType],
            "paramValues": []
        };

        for (const arg of arguments) {
            callEvent["paramValues"].push((arg || "(none)").toString());
        }

        send(JSON.stringify(callEvent));
        return targetMethod.apply(this, arguments);
    };
}

function watchMethodCall(classAndMethodName, methodParamTypes) {
    if (classAndMethodName == null || methodParamTypes == null) {
        return;
    }

    const indexOfLastSeparator = classAndMethodName.lastIndexOf(".");
    const classNamePattern = classAndMethodName.substring(0, indexOfLastSeparator);
    const methodNamePattern = classAndMethodName.substring(indexOfLastSeparator + 1);

    Java.perform(() => {
        const classOfTargetMethod = Java.use(classNamePattern);
        const possibleMethods = classOfTargetMethod[`${methodNamePattern}`];

        if (typeof possibleMethods === "undefined") {
            const failedToWatchEvent = {
                "type": "FailedToWatch",
```

(continues on next page)

(continued from previous page)

```
                "identifier": [classAndMethodName, methodParamTypes]
            };

            send(JSON.stringify(failedToWatchEvent));
            return;
        }

        possibleMethods.overloads.filter((possibleMethod) => {
            const paramTypesOfPossibleMethod = possibleMethod.argumentTypes.
↪map((argument) => argument.className);
            return paramTypesOfPossibleMethod.join(",") === methodParamTypes;
        }).forEach((matchedMethod) => {
            const retType = matchedMethod.returnType.name;
            replaceMethodImplementation(matchedMethod, classAndMethodName,␣
↪methodParamTypes, retType);
        }
        );

    });
}

rpc.exports["watchMethodCall"] = (classAndMethodName, methodParamTypes) =>␣
↪watchMethodCall(classAndMethodName, methodParamTypes);
```

2.7.3 Quark Script Result

```
$ python3 CWE-312.py
The CWE-312 vulnerability is found. The cleartext is "test@email.com"
The CWE-312 vulnerability is found. The cleartext is "password"
```

2.8 Detect CWE-89 in Android Application (AndroGoat.apk)

This scenario seeks to find SQL injection in the APK file. See CWE-89 for more details.

Let's use this APK and the above APIs to show how Quark script find this vulnerability.

First, we design a detection rule `executeSQLCommand.json` to spot on behavior using SQL command Execution. Then, we use API `isArgFromMethod` to check if `append` use the value of `getText` as the argument. If yes, we confirmed that the SQL command string is built from user input, which will cause CWE-89 vulnerability.

2.8.1 Quark Script CWE-89.py

```
from quark.script import runQuarkAnalysis, Rule

SAMPLE_PATH = "AndroGoat.apk"
RULE_PATH = "executeSQLCommand.json"

targetMethod = [
    "Landroid/widget/EditText;", # class name
    "getText",                   # method name
    "()Landroid/text/Editable;", # descriptor
```

(continues on next page)

(continued from previous page)

```
]

ruleInstance = Rule(RULE_PATH)
quarkResult = runQuarkAnalysis(SAMPLE_PATH, ruleInstance)

for sqlCommandExecution in quarkResult.behaviorOccurList:
    if sqlCommandExecution.isArgFromMethod(
        targetMethod
    ):
        print(f"CWE-89 is detected in {SAMPLE_PATH}")
```

2.8.2 Quark Rule: executeSQLCommand.json

```
{
    "crime": "Execute SQL Command",
    "permission": [],
    "api": [
        {
            "class": "Ljava/lang/StringBuilder;",
            "method": "append",
            "descriptor": "(Ljava/lang/String;)Ljava/lang/StringBuilder;"
        },
        {
            "class": "Landroid/database/sqlite/SQLiteDatabase;",
            "method": "rawQuery",
            "descriptor": "(Ljava/lang/String; [Ljava/lang/String;)Landroid/database/
↪Cursor;"
        }
    ],
    "score": 1,
    "label": []
}
```

2.8.3 Quark Script Result

```
$ python3 CWE-89.py

CWE-89 is detected in AndroGoat.apk
```

2.9 Detect CWE-926 in Android Application (dvba.apk)

This scenario seeks to find **improper export of Android application components** in the APK file. See CWE-926 for more details.

Let's use this APK and the above APIs to show how Quark script find this vulnerability.

First, we use Quark API `getActivities` to get all activity data in the manifest. Then we use `activityInstance.hasIntentFilter` to check if the activities have `intent-filter`. Also, we use `activityInstance.isExported` to check if the activities set the attribute `android:exported=true`. If both are **true**, then the APK exports the component for use by other applications. That may cause CWE-926 vulnerabilities.

2.9.1 Quark Script CWE-926.py

```
from quark.script import *

SAMPLE_PATH = "dvba.apk"

for activityInstance in getActivities(SAMPLE_PATH):

    if activityInstance.hasIntentFilter() and activityInstance.isExported():
        print(f"CWE-926 is detected in the activity, {activityInstance}")
```

2.9.2 Quark Script Result

```
$ python3 CWE-926.py

CWE-926 is found in the activity, com.app.damnvulnerablebank.CurrencyRates
CWE-926 is found in the activity, com.app.damnvulnerablebank.SplashScreen
```

2.10 Detect CWE-749 in Android Application (MSTG-Android-Java.apk)

This scenario seeks to find **exposed methods or functions** in the APK file. See CWE-749 for more details.

Let's use this APK and the above APIs to show how Quark script find this vulnerability.

First, we design a detection rule `configureJsExecution.json` to spot on behavior using method `setJavascriptEnabled`. Then, we use API `methodInstance.getArguments` to check if it enables JavaScript execution on websites. Finally, we look for calls to method `addJavaScriptInterface` in the caller method. If **yes**, the APK exposes methods or functions to websites. That causes CWE-749 vulnerability.

2.10.1 Quark Script CWE-749.py

```
from quark.script import runQuarkAnalysis, Rule

SAMPLE_PATH = "MSTG-Android-Java.apk"
RULE_PATH = "configureJsExecution.json"

targetMethod = [
    "Landroid/webkit/WebView;",
    "addJavascriptInterface",
    "(Ljava/lang/Object; Ljava/lang/String;)V"
]

ruleInstance = Rule(RULE_PATH)
quarkResult = runQuarkAnalysis(SAMPLE_PATH, ruleInstance)

for configureJsExecution in quarkResult.behaviorOccurList:

    caller = configureJsExecution.methodCaller
    secondAPI = configureJsExecution.secondAPI
```

(continues on next page)

(continued from previous page)

```
    enableJS = secondAPI.getArguments()[1]
    exposeAPI = quarkResult.findMethodInCaller(caller, targetMethod)

    if enableJS and exposeAPI:
        print(f"CWE-749 is detected in method, {caller.fullName}"
```

2.10.2 Quark Rule: configureJsExecution.json

```
{
    "crime": "Configure JavaScript execution on websites",
    "permission": [],
    "api": [
        {
            "class": "Landroid/webkit/WebView;",
            "method": "getSettings",
            "descriptor": "()Landroid/webkit/WebSettings;"
        },
        {
            "class": "Landroid/webkit/WebSettings;",
            "method": "setJavaScriptEnabled",
            "descriptor": "(Z)V"
        }
    ],
    "score": 1,
    "label": []
}
```

2.10.3 Quark Script Result

```
$ python3 CWE-749.py

CWE-749 is detected in method, Lsg/vp/owasp_mobile/OMTG_Android/OMTG_ENV_005_WebView_
↪Remote; onCreate (Landroid/os/Bundle;)V
CWE-749 is detected in method, Lsg/vp/owasp_mobile/OMTG_Android/OMTG_ENV_005_WebView_
↪Local; onCreate (Landroid/os/Bundle;)V
```

CHAPTER 3

Add Rules

Android malware analysis engine is not a new story. Every antivirus company has their own secrets to build it. With curiosity, we develop a malware scoring system from the perspective of Taiwan Criminal Law in an easy but solid way.

We have an order theory of criminal which explains stages of committing a crime. For example, crime of murder consists of five stages, they are determined, conspiracy, preparation, start and practice. The latter the stage the more we're sure that the crime is practiced.

According to the above principle, we developed our order theory of android malware. We develop five stages to see if the malicious activity is being practiced. They are

1. Permission requested.
2. Native API call.
3. Certain combination of native API.
4. Calling sequence of native API.
5. APIs that handle the same register.

We not only define malicious activities and their stages but also develop weights and thresholds for calculating the threat level of a malware.

But before we explain how to set weights and thresholds, we need to define crimes and corresponding five stages.

An example of defining crime "Send Location via SMS" is shown below. We use json format to construct the rule of the crime.

```
{
     "crime": "Send Location via SMS",

     "permission": [
         "android.permission.SEND_SMS",
         "android.permission.ACCESS_COARSE_LOCATION",
         "android.permission.ACCESS_FINE_LOCATION"
     ],
```

(continues on next page)

(continued from previous page)

```
    "api": [
    {
        "class": "Landroid/telephony/TelephonyManager",
        "method": "getCellLocation",
        "descriptor": "()Landroid/telephony/CellLocation;"
    },
    {
        "class": "Landroid/telephony/SmsManager",
        "method": "sendTextMessage",
        "descriptor": "(Ljava/lang/String; Ljava/lang/String; Ljava/lang/String;
↪Landroid/app/PendingIntent; Landroid/app/PendingIntent;)V"
    }
    ],

    "score": 4
}
```

So let me walk you through the json file.

```
"crime": "Send Location via SMS"
```

First, we define the crime. Our principle of constructing a crime is `Action + Target`. In this example, the action is "Send SMS" and the target is Location info. Therefore, the crime of our first rule is defined as: "Send Location via SMS".

```
"permission": [
      "android.permission.SEND_SMS",
      "android.permission.ACCESS_COARSE_LOCATION",
      "android.permission.ACCESS_FINE_LOCATION"
]
```

`permission` is where we fill in permission requested by the apk to practice the crime. For instance, we need permission `android.permission.SEND_SMS` to send information through SMS. We also need permission `android.permission.ACCESS_COARSE_LOCATION` and `android.permission.ACCESS_FINE_LOCATION` to practice the crime.

```
"api": [
    {
        "class": "Landroid/telephony/TelephonyManager",
        "method": "getCellLocation",
        "descriptor": "()Landroid/telephony/CellLocation;"
    },
    {
        "class": "Landroid/telephony/SmsManager",
        "method": "sendTextMessage",
        "descriptor": "(Ljava/lang/String; Ljava/lang/String; Ljava/lang/String;
↪Landroid/app/PendingIntent; Landroid/app/PendingIntent;)V"
    }
]
```

`api` means this field can be used to practice analysis from stage 2 to stage 4.

In stage 2, we need to find key native APIs that do the `Action` and `Target`. And since the API method name can be used by self-defined class. We need to fill in information of both the native API class name and method name.

Note: We like to keep our crime/rule simple. So do not fill in more than 2 native APIs.

In stage 3, we will find the combination of the native APIs we define in stage 2. Further, we will check whether they're called in the same method. If so, we will say that the combination of crime is caught! And we don't need to do anything to adjust the `api` field.

Note: We know that the native API might be wrapped in other methods. We use XREF to solve this problem.

In stage 4, we will find whether the native APIs are called in a right sequence. If so, we have more confidence that the crime is practiced.

Note: Please place the APIs in the order as the crime is being committed.

In stage 5, we will check whether the native APIs are operating the same parameter. If so, we are 100% sure that the crime is practiced.

As for the field `score`, we will be updating our principles of weight defining. please check that part later.

CHAPTER 4

Rule Generation

The Rule generation technique is based on the idea below:

1. Sort all APIs used in an APK by their usage counts.
2. Separate all APIs into two groups, P(20% least usage count) and S(other 80% APIs), by the Pareto principle (20-80 rule).
3. **Combine P and S into four different phases:**
    - PxP
    - PxS
    - SxP
    - SxS
4. Execute the rule generation with each phase in this order: PxP -> PxS -> SxP -> SxS

The earlier the phase, the higher the value of the rule but less time spent. We can generate rules in a phased manner according to different situations. For example, under a time constraint, we can take PxP phase rules as an overview for the target APK.

4.1 CLI Usage

Generate rules for APK with the following command:

```
$ quark -a <sample path> --generate-rule <generated rule directory path>
```

Generate rules and web editor with the following command:

```
$ quark -a <sample path> --generate-rule <generated rule directory path> -w <web␣
↪editor file name>
```

4.2 API Usage

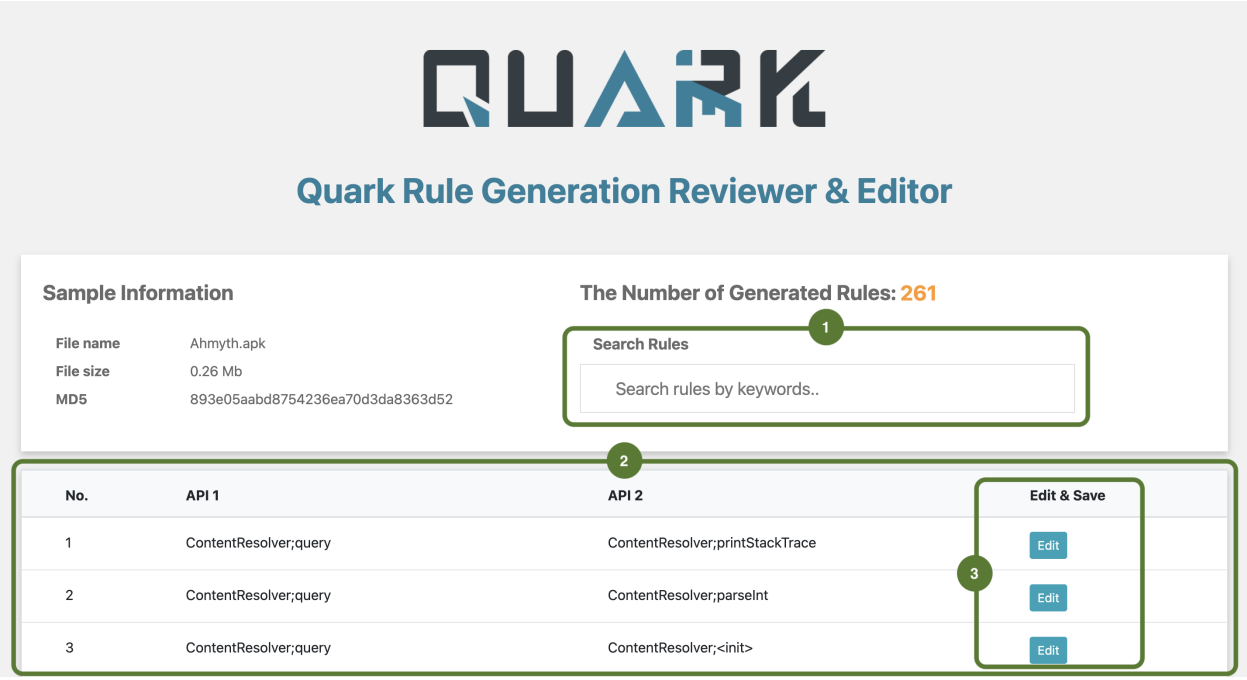
And here is the simplest way for API usage:

```
from quark.rulegeneration import RuleGeneration

# The target APK.
APK_PATH = "Ahmyth.apk"

# The output directory for generated rules.
GENERATED_RULE_DIR = "generated_rules"

generator = RuleGeneration(APK_PATH, GENERATED_RULE_DIR)
generator.generate_rule(web_editor="report.html")
```

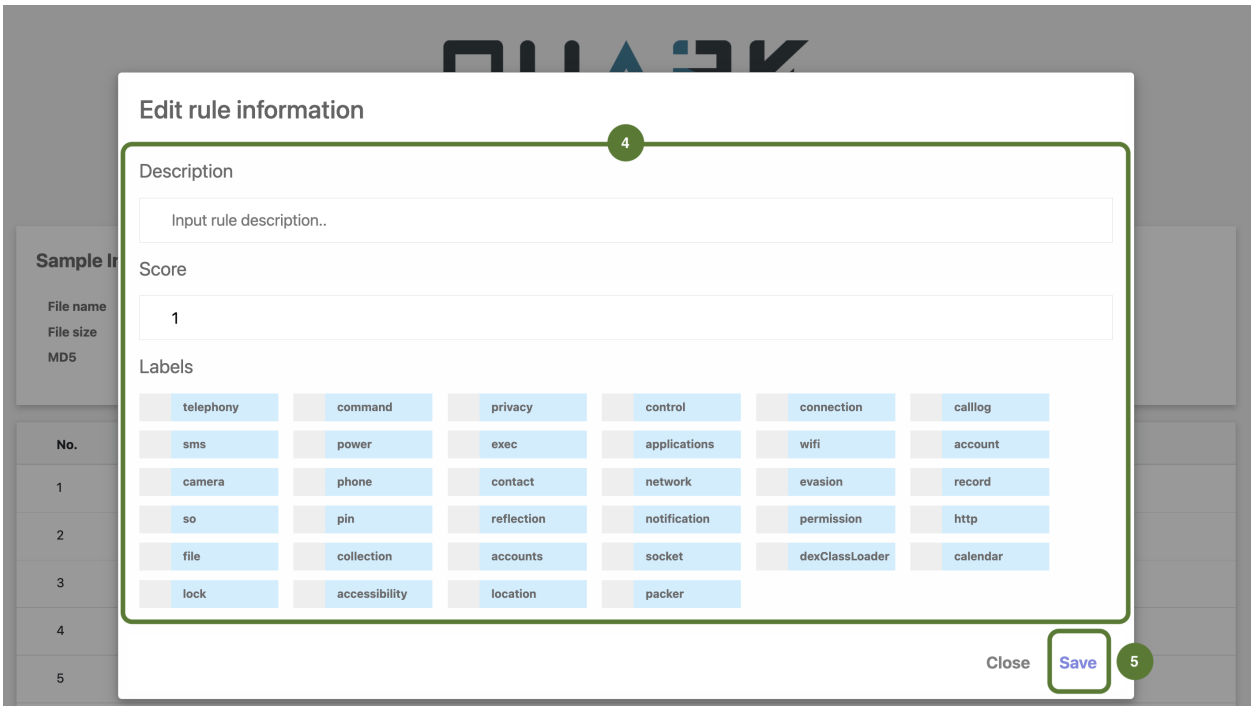
4.3 Web Editor Tutorial

Here is the demo for the rule generation web editor. You can easily review and edit generated rules with 5 steps:

1. Input keywords to search rules.
2. Select the generated rules you want to save.
3. Edit rule information.



4. Edit crime, score, and labels with the editor.
5. Save the edited rule.



4.4 Radiocontrast

Radiocontrast is a Quark API that quickly generates Quark rules from a specified method. It builds up 100% matched rules by using native APIs in that method. The feature lets you easily expose the behavior of a method, just like radiocontrast.

For example, we want to know the behavior of a method called `Lahmyth/mine/king/ahmyth/CameraManager;->startUp(I)V`, in Ahmyth.apk. Here is the simplest way for Radiocontrast usage:

```
from quark.radiocontrast import RadioContrast

# The target APK.
APK_PATH = "~/apk-malware-sample/Ahmyth.apk"

# The method that you want to generate rules.
TARGET_METHOD = "Lahmyth/mine/king/ahmyth/CameraManager;->startUp(I)V"

# The output directory for generated rules.
GENERATED_RULE_DIR = "~/generated_rules"

radiocontrast = RadioContrast(
APK_PATH,
TARGET_METHOD,
GENERATED_RULE_DIR
)

radiocontrast.generate_rule()
```

Use web editor to manage generated rules, you can define the parameter `web_editor` in `generate_rule()` as the path of output html file:

```
radiocontrast.generate_rule(web_editor="ahmyth.html")
```

The parameter `percentile_rank` in `generate_rule()` as the percentile number of API filter rank. For example, if you want to keep the 20% least usage count APIs, set the percentile_rank as 0.2:

```
radiocontrast.generate_rule(percentile_rank=0.2)
```

CHAPTER 5

Integration

Quark Engine Integration In Just 2 Steps

5.1 First Step: Installation

```
$ pip3 install -U quark-engine
```

5.2 Second Step: Code Snippet As You Go

Here we present the simplest way for quark API usage:

```
from quark.report import Report

APK_PATH = "14d9f1a92dd984d6040cc41ed06e273e.apk"
RULE_PATH = "sendLocation_SMS.json"

report = Report()

'''
RULE_PATH can be a directory with multiple rules inside
EX: "rules/"
'''
report.analysis(APK_PATH, RULE_PATH)
json_report = report.get_report("json")
print(json_report)
```

Then you get the json report. :D

```
{
    "md5": "14d9f1a92dd984d6040cc41ed06e273e",
```

(continues on next page)

(continued from previous page)


```
    "apk_filename": "14d9f1a92dd984d6040cc41ed06e273e.apk",
    "size_bytes": 166917,
    "threat_level": "High Risk",
    "total_score": 4,
    "crimes": [
        {
            "crime": "Send Location via SMS",
            "score": 4,
            "weight": 4.0,
            "confidence": "100%",
            "permissions": [
                "android.permission.SEND_SMS",
                "android.permission.ACCESS_COARSE_LOCATION",
                "android.permission.ACCESS_FINE_LOCATION"
            ],
            "native_api": [
                {
                    "class": "Landroid/telephony/TelephonyManager;",
                    "method": "getCellLocation"
                },
                {
                    "class": "Landroid/telephony/SmsManager;",
                    "method": "sendTextMessage"
                }
            ],
            "combination": [
                {
                    "class": "Landroid/telephony/TelephonyManager",
                    "method": "getCellLocation",
                    "descriptor": "()Landroid/telephony/CellLocation;"
                },
                {
                    "class": "Landroid/telephony/SmsManager",
                    "method": "sendTextMessage",
                    "descriptor": "(Ljava/lang/String; Ljava/lang/String; Ljava/lang/
↪String; Landroid/app/PendingIntent; Landroid/app/PendingIntent;)V"
                }
            ],
            "sequence": [
                {
                    "Lcom/google/progress/AndroidClientService; sendMessage ()V": {
                        "first": [
                            "invoke-virtual",
                            "v6",
                            "Lcom/google/progress/Locate;->getLocation()Ljava/lang/
↪String;"
                        ],
                        "first_hex": "6e 10 2f 02 06 00",
                        "second": [
                            "invoke-virtual",
                            "v4",
                            "v6",
                            "v7",
                            "Lcom/google/progress/SMSHelper;->sendSms(Ljava/lang/
↪String; Ljava/lang/String;)I"
                        ],
                        "second_hex": "6e 30 3e 02 64 07"
```


(continues on next page)

(continued from previous page)


```
                    }
                },
                {
                    "Lcom/google/progress/AndroidClientService; doByte ([B)V": {
                        "first": [
                            "invoke-virtual/range",
                            "v35",
                            "Lcom/google/progress/Locate;->getLocation()Ljava/lang/
↪String;"
                        ],
                        "first_hex": "74 01 2f 02 23 00",
                        "second": [
                            "invoke-virtual",
                            "v0",
                            "v1",
                            "v2",
                            "Lcom/google/progress/SMSHelper;->sendSms(Ljava/lang/
↪String; Ljava/lang/String;)I"
                        ],
                        "second_hex": "6e 30 3e 02 10 02"
                    }
                },
                {
                    "Lcom/google/progress/AndroidClientService$2; run ()V": {
                        "first": [
                            "invoke-virtual",
                            "v5",
                            "Lcom/google/progress/Locate;->getLocation()Ljava/lang/
↪String;"
                        ],
                        "first_hex": "6e 10 2f 02 05 00",
                        "second": [
                            "invoke-virtual",
                            "v3",
                            "v0",
                            "v4",
                            "Lcom/google/progress/SMSHelper;->sendSms(Ljava/lang/
↪String; Ljava/lang/String;)I"
                        ],
                        "second_hex": "6e 30 3e 02 03 04"
                    }
                }
            ],
            "register": [
                {
                    "Lcom/google/progress/AndroidClientService; sendMessage ()V": {
                        "first": [
                            "invoke-virtual",
                            "v6",
                            "Lcom/google/progress/Locate;->getLocation()Ljava/lang/
↪String;"
                        ],
                        "first_hex": "6e 10 2f 02 06 00",
                        "second": [
                            "invoke-virtual",
                            "v4",
                            "v6",
```


(continues on next page)

(continued from previous page)

```
                            "v7",
                            "Lcom/google/progress/SMSHelper;->sendSms(Ljava/lang/
↪String; Ljava/lang/String;)I"
                        ],
                        "second_hex": "6e 30 3e 02 64 07"
                    }
                },
                {
                    "Lcom/google/progress/AndroidClientService$2; run ()V": {
                        "first": [
                            "invoke-virtual",
                            "v5",
                            "Lcom/google/progress/Locate;->getLocation()Ljava/lang/
↪String;"
                        ],
                        "first_hex": "6e 10 2f 02 05 00",
                        "second": [
                            "invoke-virtual",
                            "v3",
                            "v0",
                            "v4",
                            "Lcom/google/progress/SMSHelper;->sendSms(Ljava/lang/
↪String; Ljava/lang/String;)I"
                        ],
                        "second_hex": "6e 30 3e 02 03 04"
                    }
                }
            ]
        }
    ]
}
```

5.3 Directory Scanning

To scan the entire directory with quark, you can use a simple bash script.

```
#!/bin/bash
for apkFile in *.apk; do
    quark -a ${apkFile} -o ${apkFile%%.*}_output.json;
done;
```

Alternatively, you can use the quark API as well.

```
#!/usr/bin/env python
from glob import glob

from quark.report import Report

RULE_PATH = "./quark-rules/00001.json"

report = Report()

for file in glob('*.apk'):
    report.analysis(file, RULE_PATH)
```

(continues on next page)

(continued from previous page)

```
    json_report = report.get_report("json")
    print(json_report)
```

5.4 Radiocontrast

Radiocontrast is a Quark API that quickly generates Quark rules from a specified method. It builds up 100% matched rules by using native APIs in that method. The feature lets you easily expose the behavior of a method, just like radiocontrast.

For example, we want to know the behavior of a method called *Lahmyth/mine/king/ahmyth/CameraManager;->startUp(I)V,* in Ahmyth.apk. Here is the simplest way for Radiocontrast usage:

```
from quark.radiocontrast import RadioContrast

# The target APK.
APK_PATH = "~/apk-malware-sample/Ahmyth.apk"

# The method that you want to generate rules.
TARGET_METHOD = "Lahmyth/mine/king/ahmyth/CameraManager;->startUp(I)V"

# The output directory for generated rules.
GENERATED_RULE_DIR = "~/generated_rules"

radiocontrast = RadioContrast(
    APK_PATH,
    TARGET_METHOD,
    GENERATED_RULE_DIR
)
radiocontrast.rule_generate()
```

CHAPTER 6

Development

This chapter explains how to write Quark's code and how to contribute.

6.1 Development Notes

6.1.1 Git Branching Model

Quark-engine source code is available in our official Git repository.

We have 6 branches.

- **master**:
  - This is the stable and releasing branch.
- **dev**:
  - This is where we commit our beta version.
  - When all the features are fully tested, it will be merged to `master` branch.
- **feature**:
  - This is where we develop new features for the project.
  - Once it's done, it will be merged to `dev` branch.
- **testing**:
  - This is where we write tests for the codebase and new feature.
  - When it's done, it will be merged to `dev` branch.
- **hotfix**:
  - This branch is used to fix bugs.
  - Will be merged to `dev` and `master`

- **doc**:
  - This is where we update our documentation.
  - Just like hotfix branch, will be merged to `dev` and `master`.

6.1.2 Release Versioning

Our versioning logic is quite simple. We use the year and month of the release as a version number. `v19.10` means we release this version in Oct. 2019.

6.1.3 Ticketing System

To submit reports or feature requests, please use GitHub's Issue tracking system.

6.1.4 Contribute

To submit your patch, just create a Pull Request from your GitHub fork. If you don't know how to create a Pull Request take a look to GitHub help.

6.2 Coding Style

Having a clean and structured code is very important for the sustainable development of this project. Cuckoo Sandbox is a good example. So we follow most of their style for maintaining the code base.

We do help out with code refactoring where required, but please try to do as much as possible on your own.

Essentially Quark's code style is based on PEP 8 - Style Guide for Python Code and PEP 257 – Docstring Conventions.

6.2.1 Formatting

File and folder naming

All characters in the filenames or folder names should be lowercase letters.

Copyright header

All existing source code files start with the following copyright header:

```
# This file is part of Quark Engine - https://quark-engine.rtfd.io
# See GPLv3 for copying permission.
```

Indentation

The code must have a 4-spaces-tabs indentation. Since Python enforce the indentation, make sure to configure your editor properly or your code might cause malfunctioning.

Maximum Line Length

Limit all lines to a maximum of 79 characters.

Blank Lines

Separate the class definition and the top level function with one blank line. Methods definitions inside a class are separated by a single blank line:

```
class MyClass:
    """Doing something."""

    def __init__(self):
        """Initialize"""
        pass

    def do_it(self, what):
        """Do it.
        @param what: do what.
        """
        pass
```

Use blank lines in functions, sparingly, to isolate logic sections. Import blocks are separated by a single blank line, import blocks are separated from classes by one blank line.

Imports

Imports must be on separate lines. If you're importing multiple objects from a package, use a single line:

```
from lib import a, b, c
```

NOT:

```
from lib import a
from lib import b
from lib import c
```

Always specify explicitly the objects to import:

```
from lib import a, b, c
```

NOT:

```
from lib import *
```

Strings

Strings must be delimited by double quotes (").

Printing and Logging

We discourage the use of `print()`: if you need to log an event please use Python's `logging` which is already initialized by Quark.

In your module add:

```
import logging
log = logging.getLogger(__name__)
```

And use the `log` handle. More details can be found in the Python documentation, but as follows is an example:

```
log.info("Log message")
```

6.2.2 Exceptions

Custom exceptions will be defined in the `quark/utils/exceptions.py` file.

Naming

Custom exception names must start with "Quark" and end with "Error" if it represents an unexpected malfunction.

Exception handling

When catching an exception and accessing its handle, use `as e`:

```
try:
    foo()
except Exception as e:
    bar()
```

NOT:

```
try:
    foo()
except Exception, something:
    bar()
```

It's a good practice use "e" instead of "e.message".

6.2.3 Documentation

All code must be documented in docstring format, see PEP 257 – Docstring Conventions. Additional comments may be added in logical blocks to make the code easier to understand.

As for the contribution to quark rtdf. Please use `make html` to build the html files. And commit with rst files altogether. So that users can read docs when offline.

6.2.4 Automated testing

We believe in automated testing to provide high quality code and avoid easily overlooked mistakes.

When possible, all code must be committed with proper unit tests. Particular attention must be placed when fixing bugs: it's good practice to write unit tests to reproduce the bug. All unit tests and fixtures are placed in the tests folder in the quark-engine/tests.

We have adopted Pytest as unit testing framework. Also we adopted travis ci for the continuous integration.

CHAPTER 7

Contribution

7.1 Different stages of core members

- `Contributor`: As soon as you post a comment, fix a typo, ask a question or report a bug etc. , you become an active contributor.
- `Contributor followed by a mentor`: Spotted active contributors can be asked to get a mentor to speedup their learning of this project.
- `Core member`: Once other core members consider that a contributor is ready to be promoted, a core member opens a private vote for the candidate.

7.2 Core members

Name	Contribution
KunYu Chen	`BDFL, Project Founder, The man who runs the show`
YuShiang Dang	`UI/UX Team Lead, Core Developer, Rule generation system, Malware Analyst`
ShengFone Lu	`Core Developer, Triage Team Lead, Rizin Engine Implementation`
AnWei Kung	`Core Developer, Triage Team Member, Backend (database, downloader, API), Testing`
Zin Wong	`Triage Team Member`
Zee	`UI/UX Team Member`

7.3 Alumni

Name	Contribution
JunWei Song	`Project Co-Founder, Engine Implementation (Malware Scoring System), Frontend, Testing`
ShunTe Lin	`Core Developer, Triage Team Member`
ChaoWen Li	`Quark Community Development`
Sin	`Malware Analyst, Detection Rules, Engine Implementation (Malware Anti-Detection)`
IokJin Sih	`Malware Analyst`
YunChen Liu	`Artwork Design, Frontend`

7.4 Consultants

Consultant is the one who is important to this project and shows great support to our engine.

Consultant
Chun-I Fan
HY Lin
Jheng-Jia Huang
Lin Gao-Yu
Pippen Wang

CHAPTER 8

Organization

8.1 Quark triage team

The Quark triage team handles more than just triage. They are the managers of the Quark organization. They define processes for handling issues/PRs, manage Quark releases. And the most important of all, they communicate with community members.

8.1.1 Responsibilities

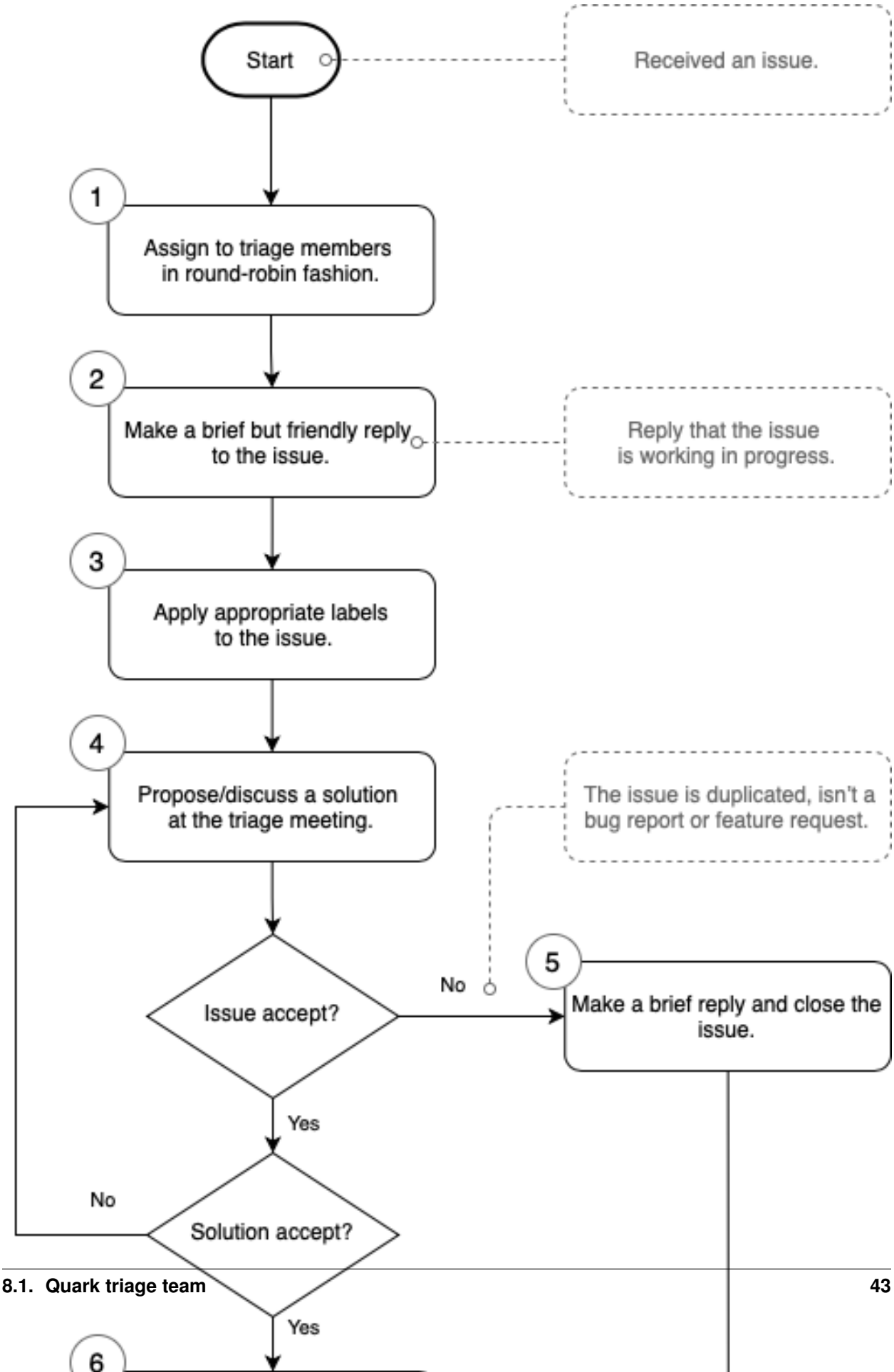
`Version: v1.0`

- **Management of issues within the Quark Github organization, including:**
  - Review, close issues and apply appropriate labels for issues.
  - Resolve minor issues (typo, usage problem, ask about the Quark's techniques).
  - Initiate a discussion with other members if the issues are major changes.
  - Define and revise the process for handling issues.
- **Management of PRs within the Quark Github organization, including:**
  - Review, merging PRs and applying appropriate labels for PRs.
  - Initiate a discussion with other members if the PRs are major changes.
  - Define and revise the process for handling PRs.
- **Management of Quark releases on Kali/PyPi/Github, including:**
  - Generate and create releases.
  - Test releases.
  - Define and revise the process and determine the content for releases.

8.1.2 Triage process

`Version: v1.1`

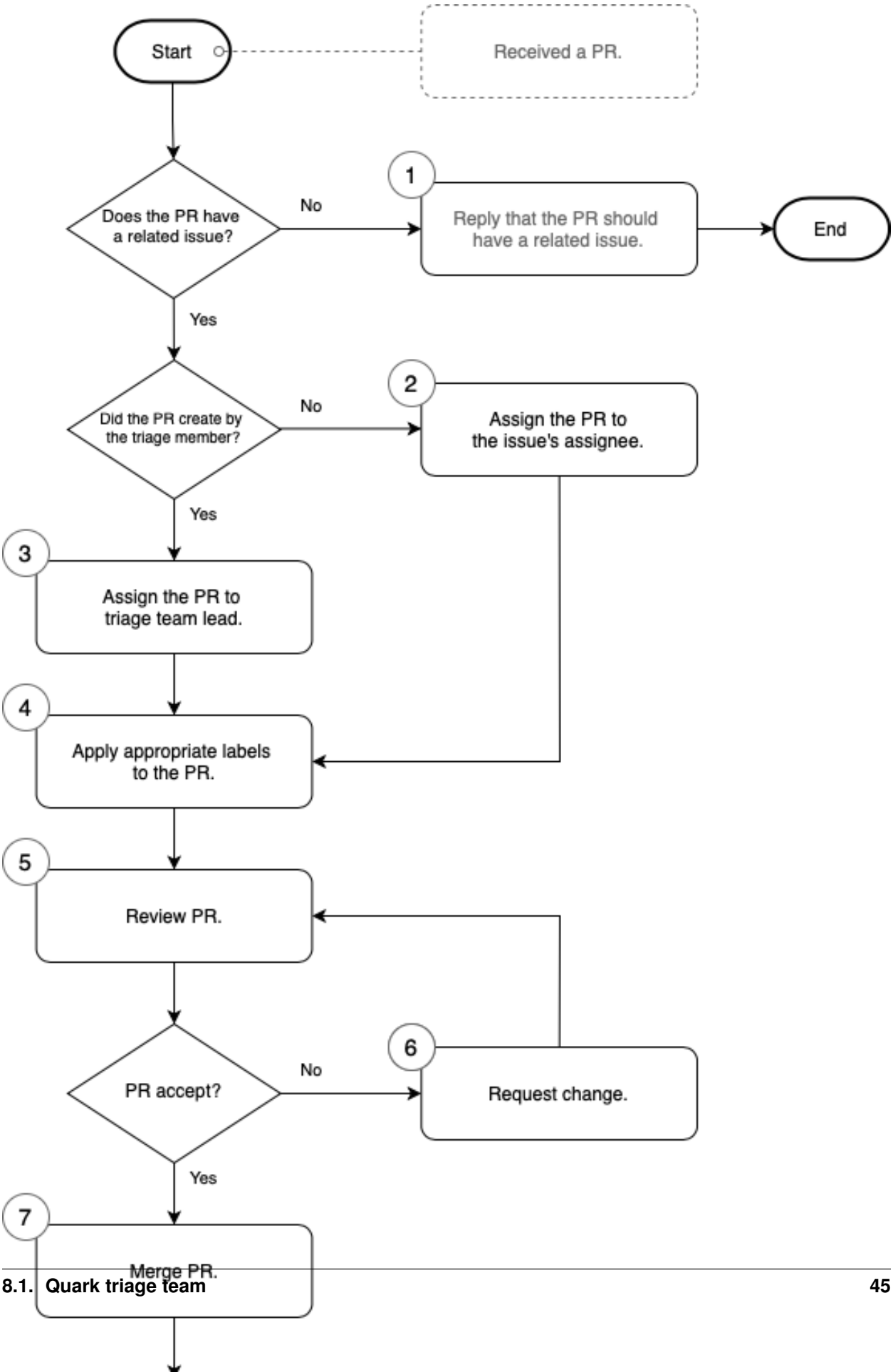
When Quark received an issue, the triage process is as below:


Start
Received an issue.
1
Assign to triage members in round-robin fashion.
2
Make a brief but friendly reply to the issue.
Reply that the issue is working in progress.
3
Apply appropriate labels to the issue.
4
Propose/discuss a solution at the triage meeting.
The issue is duplicated, isn't a bug report or feature request.
Issue accept?
No
5
Make a brief reply and close the issue.
Yes
No
Solution accept?
Yes
6

Note: If the issue is unable to reproduce, and the issue raiser hasn't responded.

- **Over one month**, the assignee should apply the label `more-info-require` to the issue and ask for more information.
- **Over two months**, the assignee should apply the label `invalid` to the issue and close it.

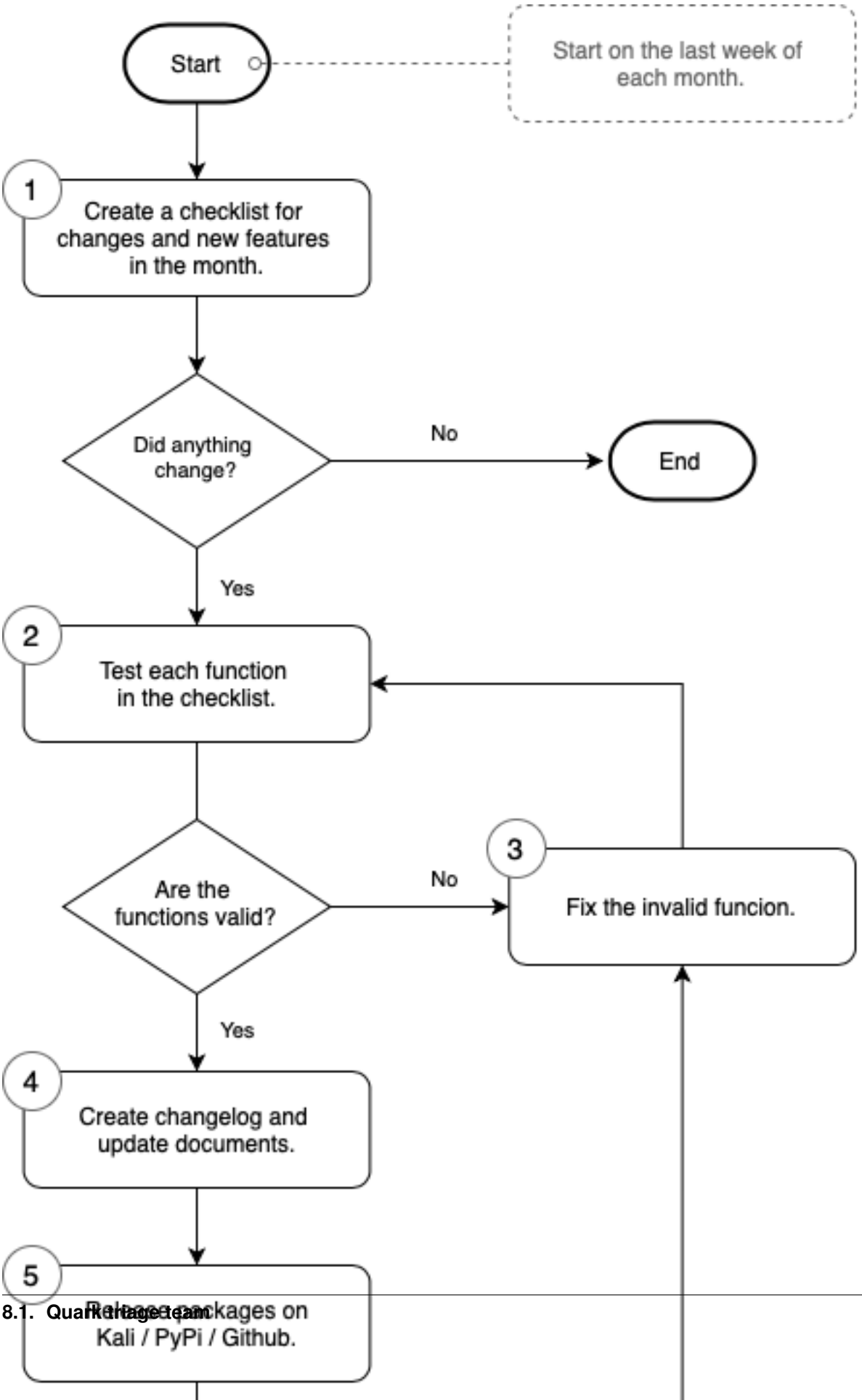
When Quark received a PR, the triage process is as below:


Start
Received a PR.
Does the PR have a related issue?
No
1
Reply that the PR should have a related issue.
End
Yes
Did the PR create by the triage member?
No
2
Assign the PR to the issue's assignee.
Yes
3
Assign the PR to triage team lead.
4
Apply appropriate labels to the PR.
5
Review PR.
PR accept?
No
6
Request change.
Yes
7
Merge PR.

8.1.3 Release process

`Version: v1.0`

The Quark release process is as below:


Start
Start on the last week of each month.
1
Create a checklist for changes and new features in the month.
Did anything change?
No
End
Yes
2
Test each function in the checklist.
Are the functions valid?
No
3
Fix the invalid funcion.
Yes
4
Create changelog and update documents.
5
Release packages on Kali / PyPi / Github.

CHAPTER 9

Updating Documentation

If you want to help update Quark documentation, you can refer to the following command to update. To make documentation can be read offline, we will reserve the HTML files for users.

After editing the docs you would like to modify, you can type the following commands to make quark generate documentation automatically.

Inside the directory of `quark-engine/docs/`:

Clean the source:

```
$ make clean
```

Auto gen the doc:

```
$ sphinx-apidoc -o source ../quark -f
```

Build the HTML:

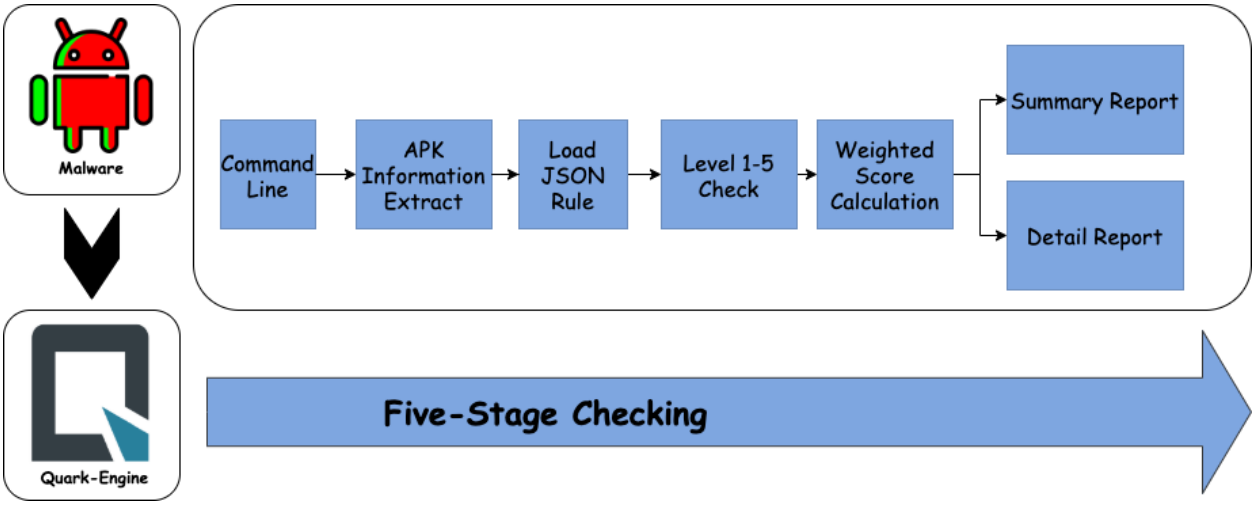
```
$ make html
```

CHAPTER 10

Quark-Engine Inside

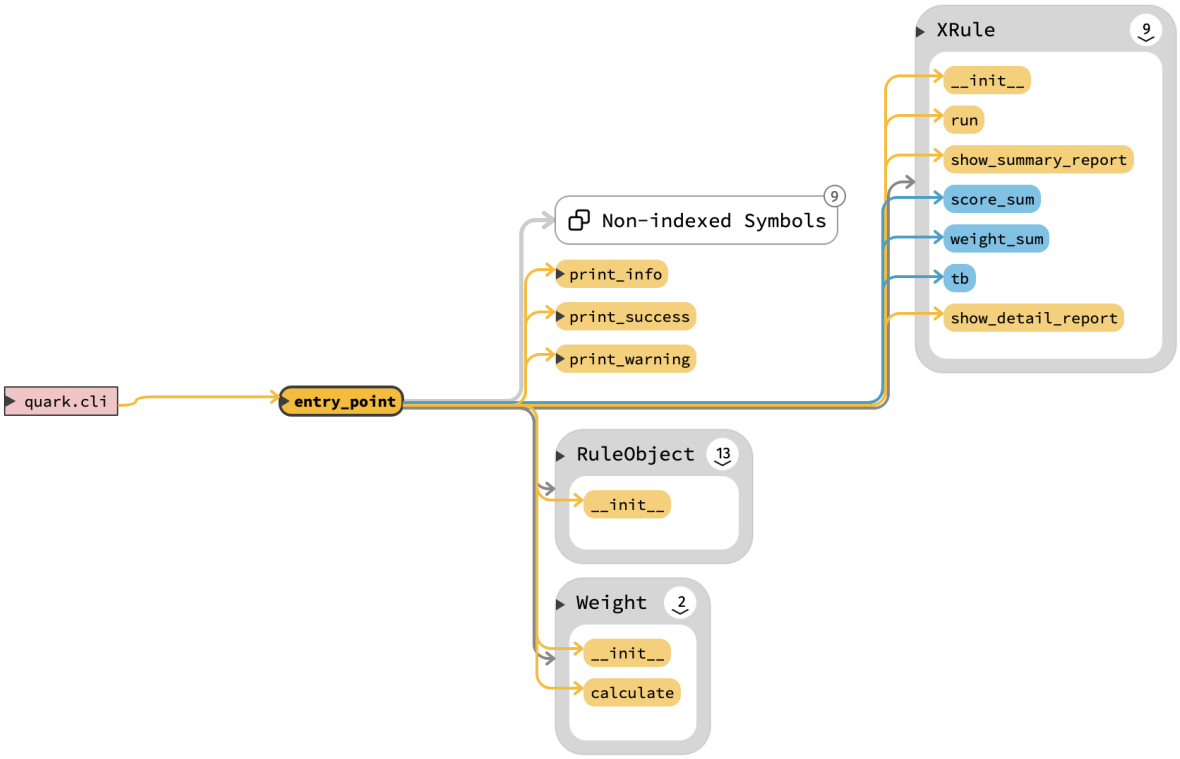
This chapter explains how Quark work inside.

10.1 Quark-Engine Workflow



10.1.1 Quark-Engine Step 1(Command Line)

`quark.cli` is the entry point of the program, and it will initialize the `XRule` object and the `RuleObject` object according to the given APK file and JSON rules, and create `quark.utils.weight` object at the end to calculate the weighted score, finally, display the report according to `-s` or `-d`. No matter whether you choose the detail report or the summary report, a full analysis will be run. The difference is that the report display is different.



Quark-Engine through the command interface to execute malware analysis like below:

Summary Report:

```
$ quark -a malware.apk -s
```

Detail Report:

```
$ quark -a malware.apk -d
```

The Quark-Engine will start from the `quark.cli` module, which is our first step in the above image.:

```
-a specifies an apk file
-r will specify a rule directory
-s for summary report
-d for detail report
```

10.1.2 Quark-Engine Step 2 (APK Information Extract)

In step 2, we will extract the information we want from the given APK file, such as the permission request list, what native APIs are called, and with the help of androguard, we can find the cross-reference method from the given function name, and also get the Dalvik bytecode instruction.

10.1.3 Quark-Engine Step 3 (Load JSON Rule)

In step 3, we will traverse each JSON file from the rules folder given by `-r` in the command-line interface, and each JSON file will be considered a five-stage rule of malicious behavior.

10.1.4 Quark-Engine Step 4 (Level 1-5 Check)

In step 4, We will follow our custom five-stage crime rules, which are as follows:

1. Permission requested.
2. Native API call.
3. Certain combination of native API.
4. Calling sequence of native API.
5. APIs that handle the same register.

Detailed implementation principle, please refer to the `quark.Objects.xrule`.

10.1.5 Quark-Engine Step 5 (Weighted Score Calculation)

In step 5, We will calculate the weighted score of each five-stage crime rule based on the stages we found by each rule, and sum up each score. Further, we will have a set of formulas to evaluate which risk range this weighted score is, such as `low risk`, `medium risk`, and `high risk`.

First of all, in each of the five-stage crime rules, there is a field called score. This score is based on the severity of the crime.

Take this rule for example:

```
{
    "crime": "Send Location via SMS",

    "permission": [
        "android.permission.SEND_SMS",
        "android.permission.ACCESS_COARSE_LOCATION",
        "android.permission.ACCESS_FINE_LOCATION"
    ],

    "api": [
        {
            "class": "Landroid/telephony/TelephonyManager",
            "method": "getCellLocation"
        },
        {
            "class": "Landroid/telephony/SmsManager",
            "method": "sendTextMessage"
        }
    ],

    "score": 4
}
```

As you can see, this rule has a field called `score`, which is `4`. Then, after the `RuleObject` initialize this rule, you can get this score by using `RuleObject.yscore`.

After that, while the five-stage analysis check is completed, we can know which crime stages have been reached. We will use `get_score` to get the weighted score of this five-stage crime rule.

```
def get_score(self, confidence):
    """
    According to the state of the five stages, we calculate the weighted score based␣
↪on exponential growth.
```

(continues on next page)

(continued from previous page)

```
    For example, we captured the third stage in five stages, then the weighted score would be (2^3-1) / 2^4.

    2^(confidence - 1)

    :param confidence:
    :return: floating point
    """
    if confidence == 0:
        return 0
    return (2 ** (confidence - 1) * self._score) / 2 ** 4
```

So assuming this rule, we captured the `fourth stage`, that is, we can confirm that the two native APIs appear in order. Then the calculation of this score is (2 ** (4 - 1) * self._score) / 2 ** 4, which is 2.

As for our risk range is defined in function `calculate` of `qaurk.utils.weight`.

There are five level threshold, the range are defined as below:

```
# Level 1 threshold
level_one_threshold = self.score_sum / 2 ** 4

# Level 2 threshold
level_two_threshold = self.score_sum / 2 ** 3

# Level 3 threshold
level_three_threshold = self.score_sum / 2 ** 2

# Level 4 threshold
level_four_threshold = self.score_sum / 2 ** 1

# Level 5 threshold
level_five_threshold = self.score_sum / 2 ** 0
```

If the final total risk score falls in the `first and second` stages, it is **low risk**; if it is in the `third and fourth` stages, it is **medium risk**. If it is in the `fifth` stage, it is **high risk**.

10.1.6 Quark-Engine Step 6 (Report)

As a final step, we present our analysis report in two forms, a summary report and a detailed report.

Summary Report

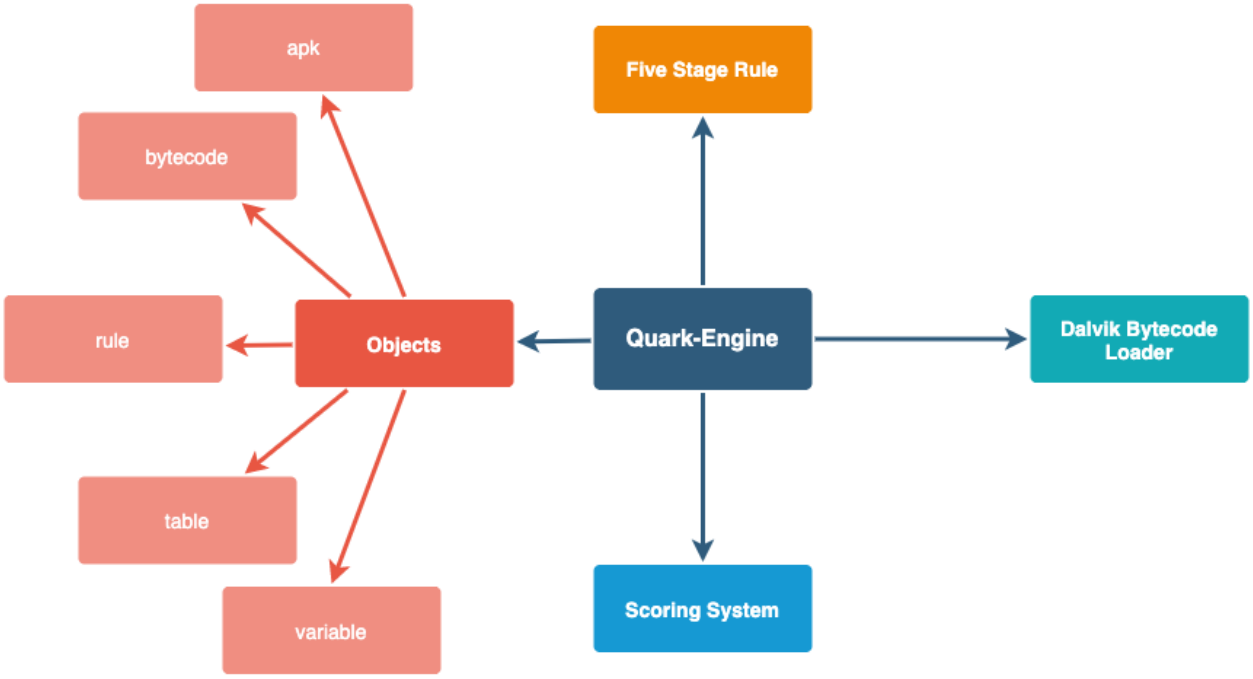
```
    _______                        __
    \_____  \  __ _______ ______ |  | __
     /  / \  \|  |  \__  \_  __ \  |/ /
    /   \_/.  \  |  // __ \|  | \/    <
    \_____\ \_/____/(____  /__|  |__|_ \
          \__>          \/           \/ v20.01

            An Obfuscation-Neglect Android Malware Scoring System

100%|████████████████████████████████████████████████████████████████████████| 8/8 [00:00<00:00,  9.06it/s]
[!] WARNING: Moderate Risk
[*] Total Score: 29
+---------------------------+------------+-------+---------+
| Rule                      | Confidence | Score | Weight  |
+---------------------------+------------+-------+---------+
| Send file via socket      | 20%        | 6     | 0.375   |
| Send recording via socket | 60%        | 5     | 1.25    |
| Send contact via socket   | 60%        | 2     | 0.5     |
| Send file via SMS         | 20%        | 6     | 0.375   |
| Send Location via SMS     | 100%       | 4     | 4.0     |
| Send contact via SMS      | 100%       | 2     | 2.0     |
| Send file via SMS         | 20%        | 3     | 0.1875  |
| Delete SMS                | 80%        | 1     | 0.5     |
+---------------------------+------------+-------+---------+
```

Detail Report

```
    _______                        __
    \_____  \  __ _______ ______ |  | __
     /  / \  \|  |  \__  \_  __ \  |/ /
    /   \_/.  \  |  // __ \|  | \/    <
    \_____\ \_/____/(____  /__|  |__|_ \
          \__>          \/           \/ v20.01

            An Obfuscation-Neglect Android Malware Scoring System

  0%|                                                                                | 0/1 [00:00<?, ?it/s]
rules/sendLocation_SMS.json

Confidence: 100%

        [✓]1.Permission Request
                android.permission.SEND_SMS
                android.permission.ACCESS_COARSE_LOCATION
                android.permission.ACCESS_FINE_LOCATION
        [✓]2.Native API Usage
                getCellLocation
        [✓]3.Native API Combination
                getCellLocation
                sendTextMessage
        [✓]4.Native API Sequence
                Sequence show up in:
                (Lcom/google/progress/AndroidClientService;, doByte)
                (Lcom/google/progress/AndroidClientService;, sendMessage)
        [✓]5.Native API Use Same Parameter
                (Lcom/google/progress/AndroidClientService;, sendMessage)
[+] DONE: OK
```

10.2 Quark-Engine Project Overview

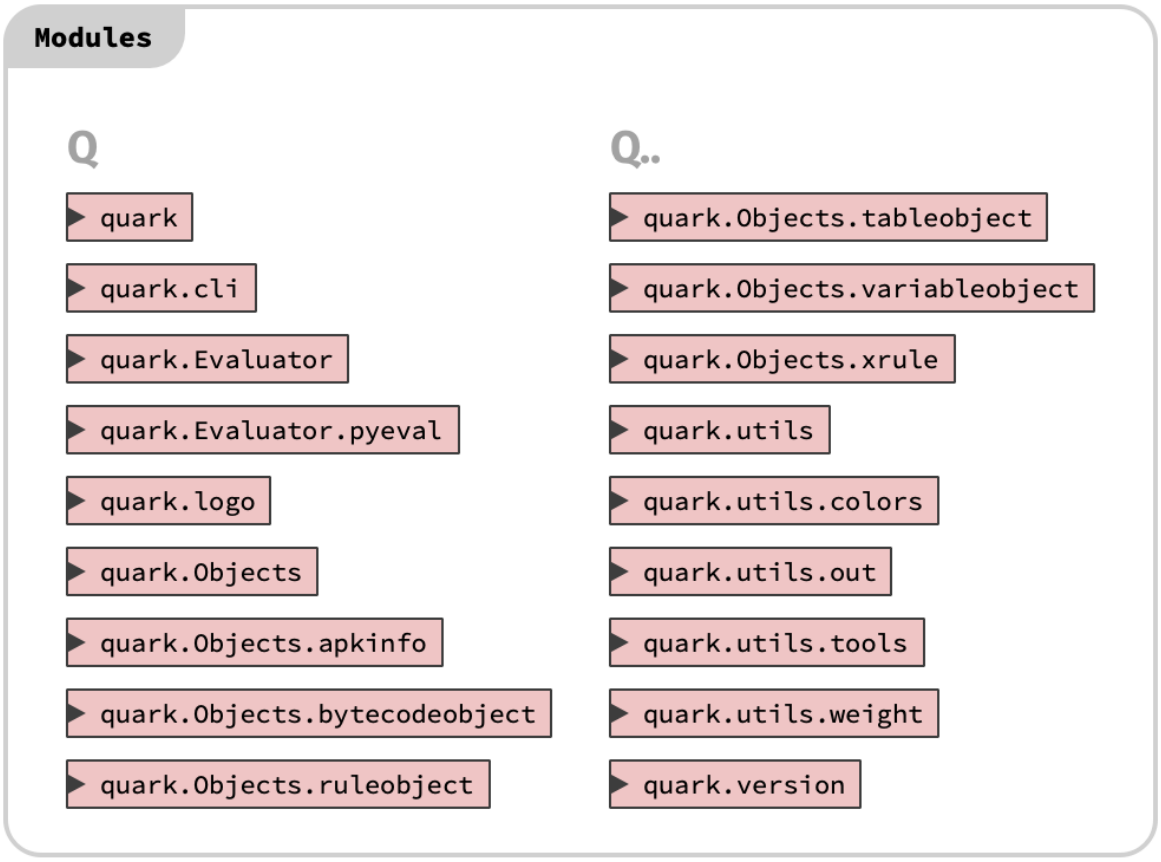
10.2.1 Quark-Engine Map



The entire Quark-Engine project can be composed of four major parts, namely

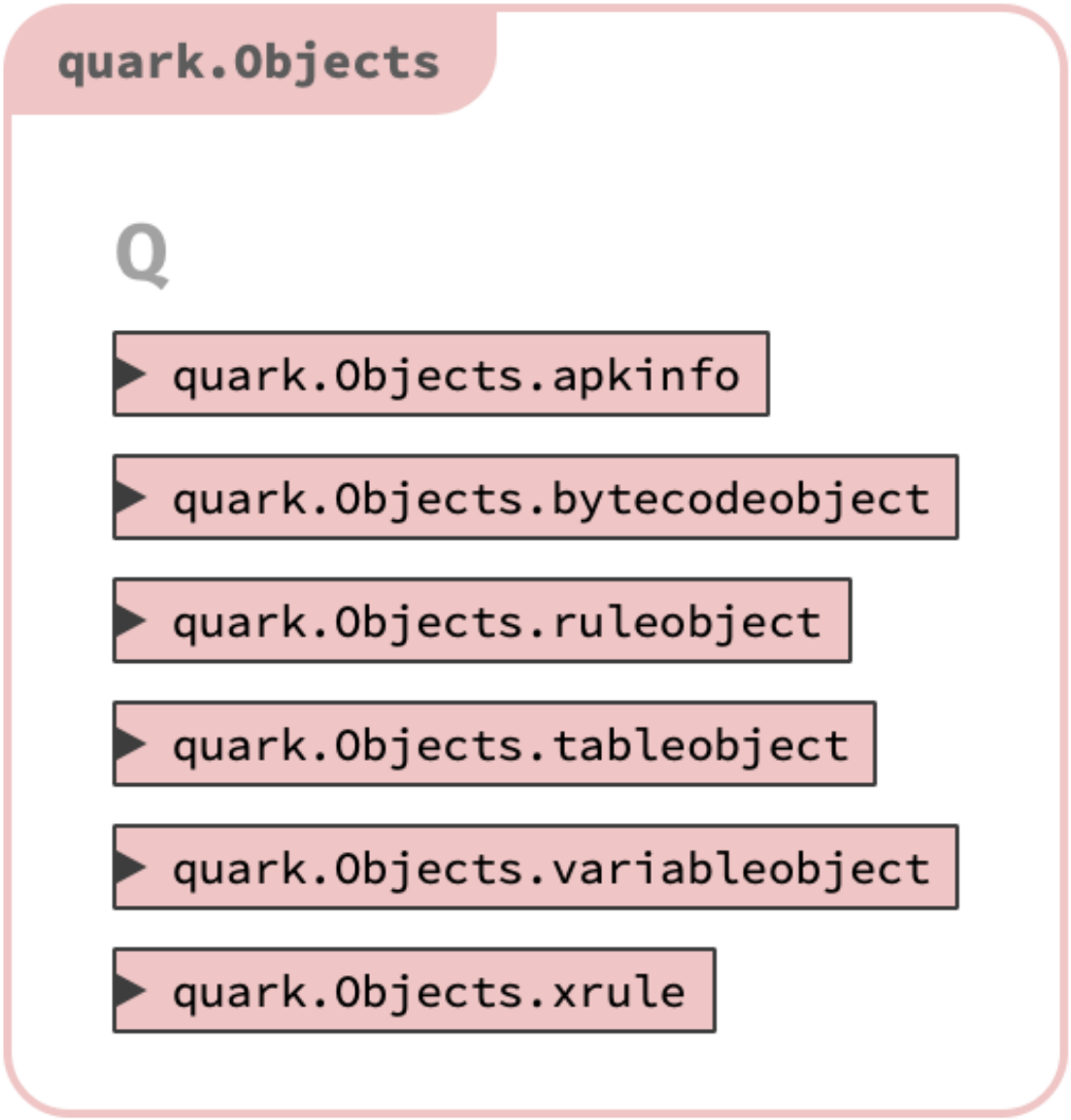
- Five-stage rule
- Dalvik Bytecode Loader
- Scoring System
- Self-define objects data structure

10.2.2 Quark module architecture diagram



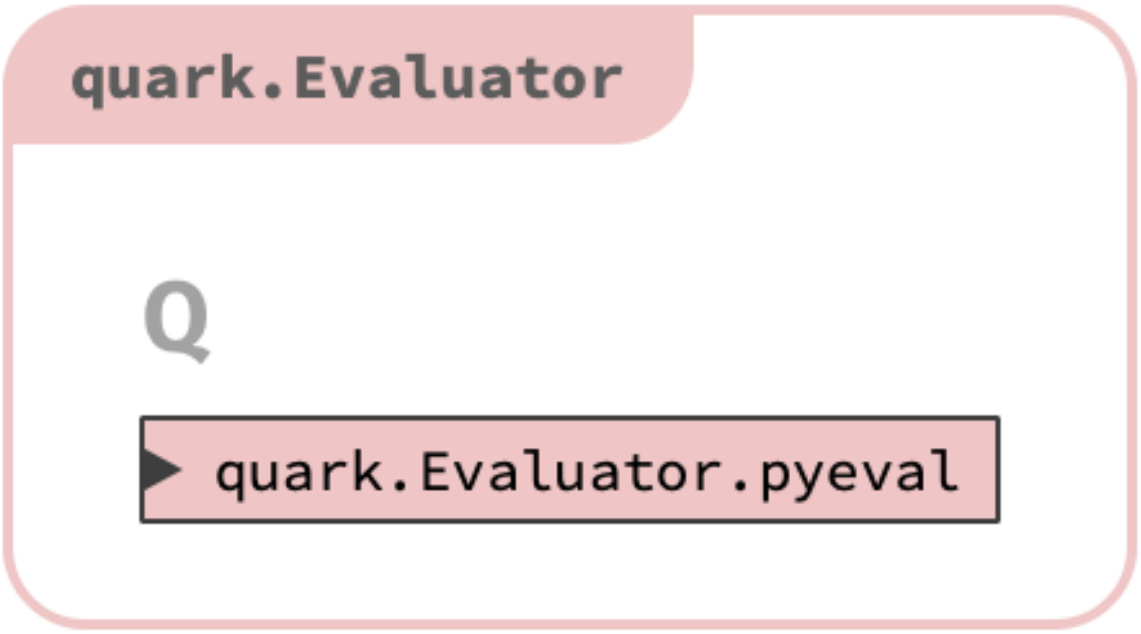
The project is divided into three main folders.

Objects



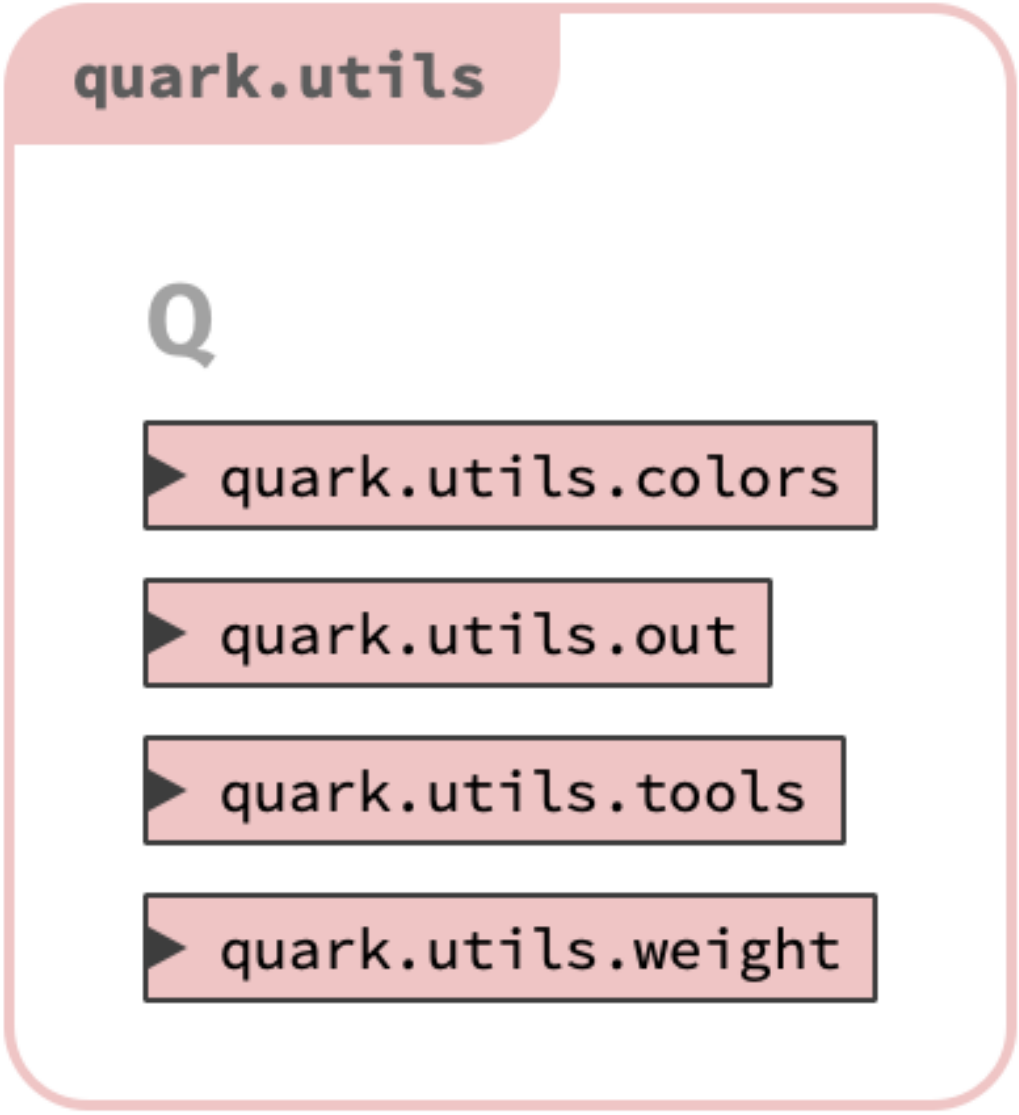
The Objects directory is used to store all the main self define objects, including APK information object, Bytecode object, rule object, variable tracking table object, variable object, and five-stage rule object.

Evaluator



The Evaluator directory is used to store the Dalvik Bytecode Loader. The name comes from the CPython virtual machine used to execute the python bytecode. The Bytecode Loader itself is a huge switch. When the corresponding Bytecode instruction is given, our customized function event will be executed. However, the bytecode instruction does not interact with the CPU, so it is faster than executing Android DEX files dynamically.

utils



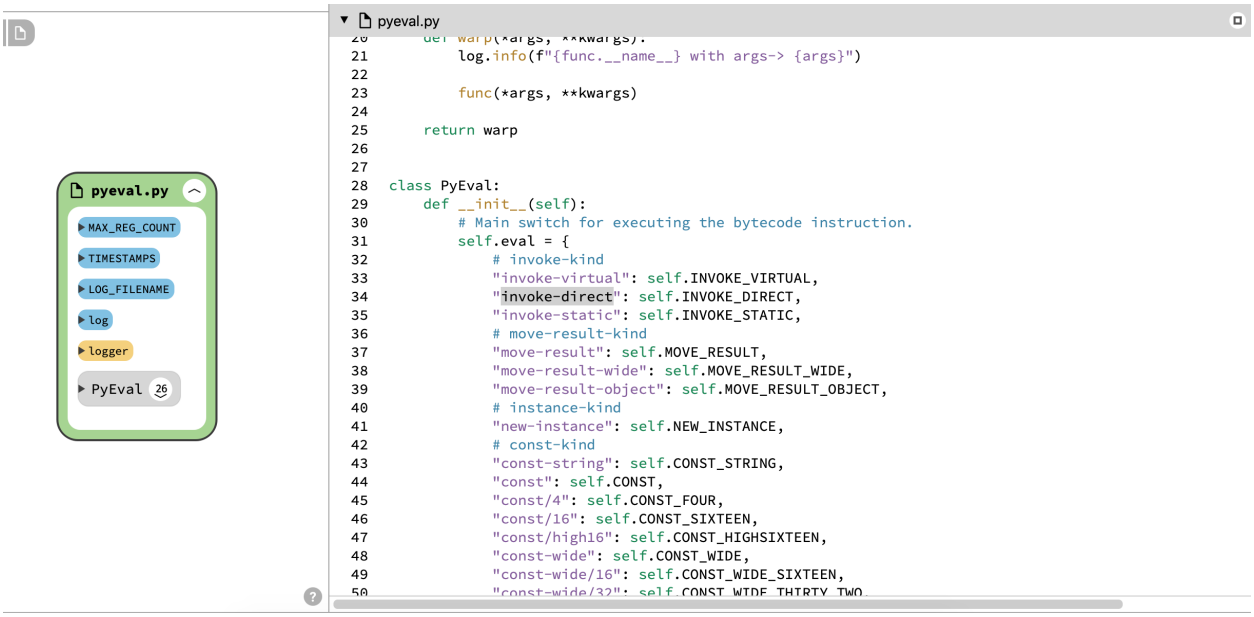
The utils directory is used to store repetitive tool code, print output control, and weighted score calculations.

10.3 Quark-Engine Objects Introduction

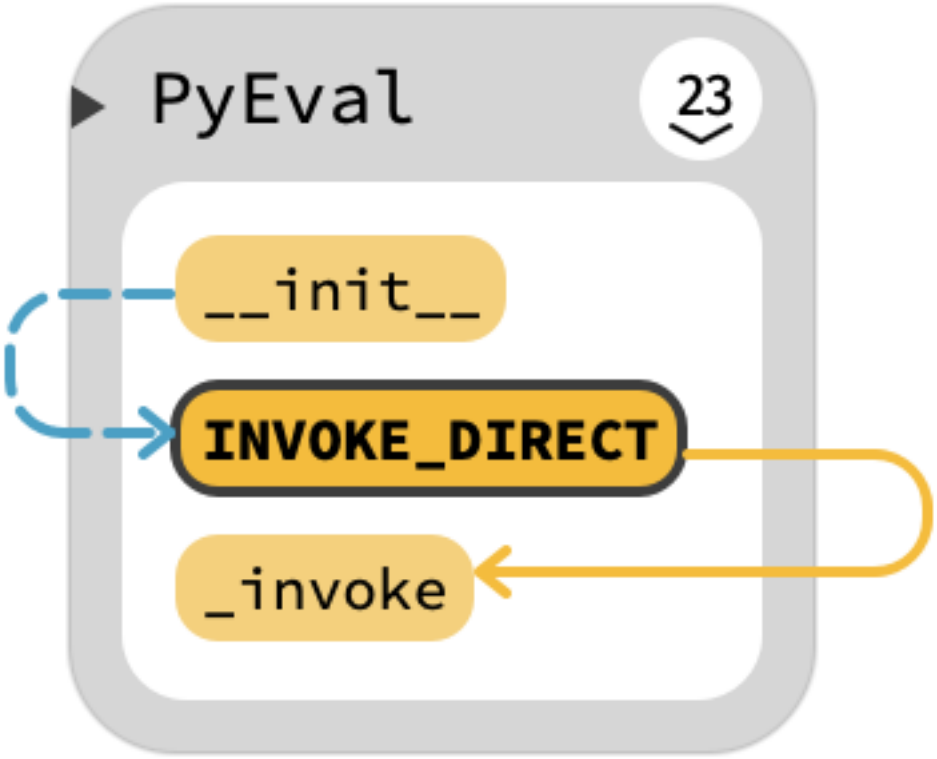
10.3.1 Dalvik Bytecode Loader(quark.Evaluator.pyeval)

PyEval, inspired by the `ceval.c` design principle in CPython's Interpreter. It takes python bytecode instructions into

an infinite loop and interacts with the CPU using C language. Thus, we apply this principle to our Dalvik Bytecode Loader, takes Dalvik's Bytecode instructions, and apply our custom instructions events to implement tracking whether two function calls operating the same variable in the fifth crime stage. Of course, we haven't implemented all the bytecode instructions, and we haven't considered the conditional jump now.

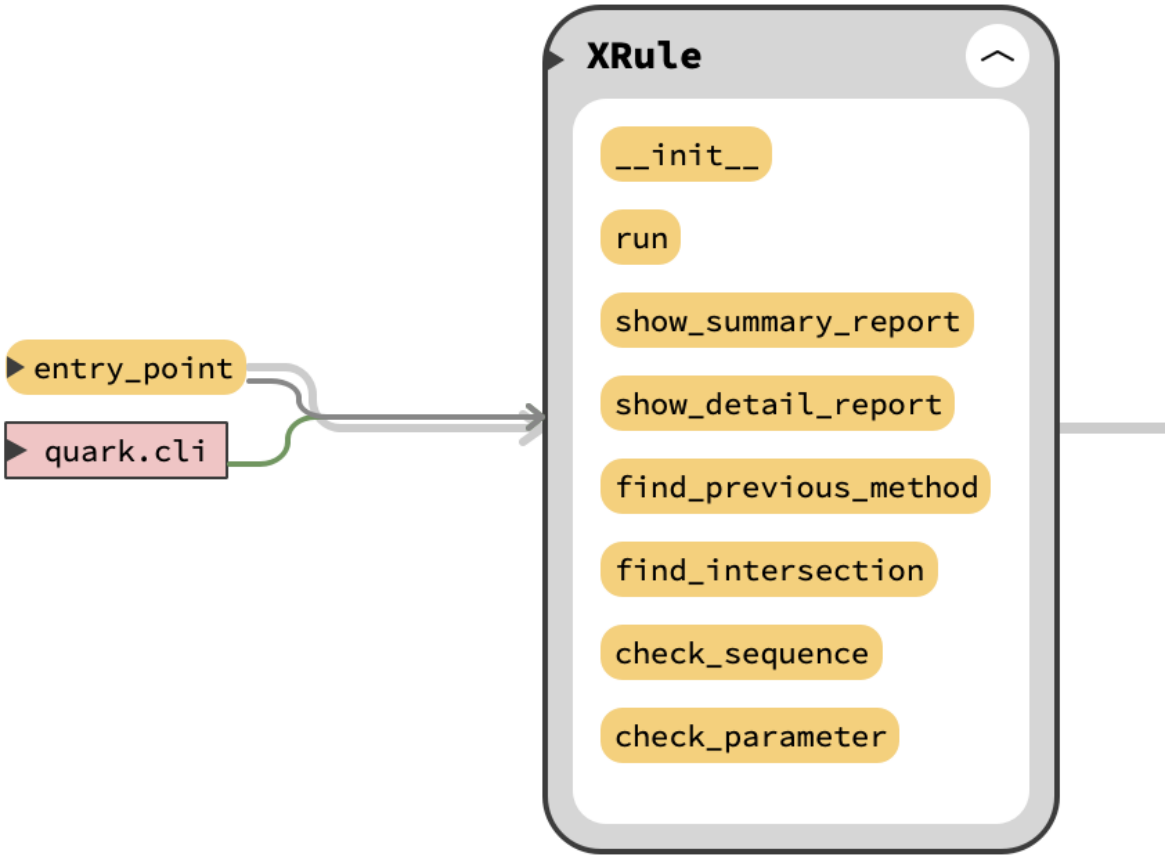


Take Android's bytecode instruction `invoke-direct` as an example. When an instruction `invoke-direct` is passed to our Dalvik bytecode loader, it will enter the PyEval main switch, execute the corresponding `INVOKE_DIRECT` function, and call `self._invoke()`, here another `_invoke` is written for reuse, because there are many instructions related to `invoke` family, such as `invoke-direct`, `invoke-virtual`, but the invoke family is the same for our program implementation.



10.3.2 XRule(quark.Objects.XRule)

XRule as an object in the quark-engine responsible for the five-stage inspection, each APK will initialize an XRule object respectively.



Explanation of each function

- **run**:
  - This function responsible for starting the five-stage inspection.
- **show_summary_report**:
  - Show the summary report.
- **show_detail_report**:
  - Show detailed report.
- **find_previous_method**:
  - Track the previous function of a specific function.
- **find_intersection**:
  - Find if there is a function of intersection in a given two functions.
- **check_sequence**:
  - Check if the two function is in order or not.
- **check_parameter**:

- Check if the two functions operate on the same variable or not.

Five-stage inspection

Each function in this XRul object is used to correspond to the five-stage inspection, which will be described in order according to the five-stage inspection.

Level 1

Check whether all permission requirements match the given rules.

Level 2

Check whether all native API functions match the given rules (as long as one of them is met).

Level 3

Check if the native API functions all match to the given rules (both appear).

Level 4

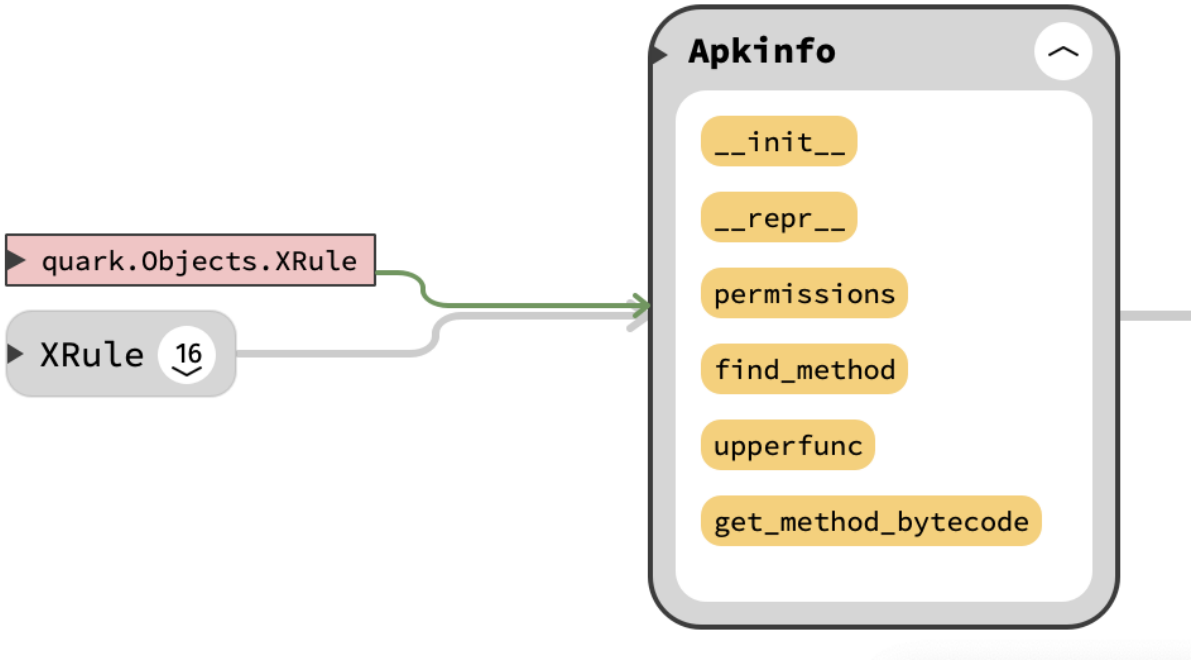
Check if the native API functions all match to the given rules (both appear in the order).

Level 5

Check whether the native API functions all match to the given rules (both appear in the order, and operate on the same variable).

10.3.3 Apkinfo(quark.Objects.Apkinfo)

Apkinfo is an object in Quark-Engine used to store APK information which is based on `androguard` module.

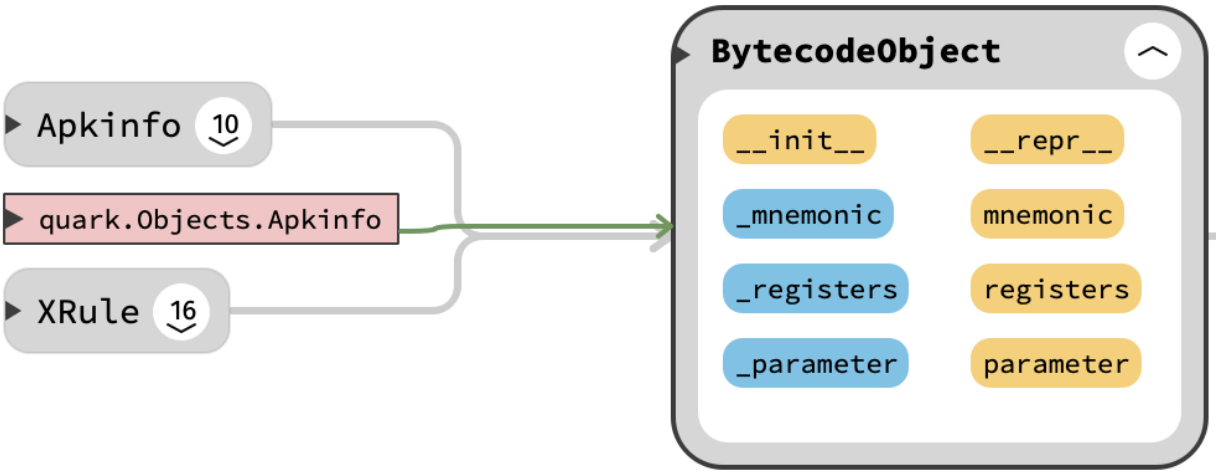


Explanation of each function

- **permissions**:
  - Get all permissions required by the APK.
- **find_method**:
  - Query function name in APK.
- **upperfunc**:
  - Query the upper-level function call for a given function name in the APK.
- **get_method_bytecode**:
  - Returns the Android bytecode corresponding to the given function name.

10.3.4 BytecodeObject(quark.Objects.bytecodeObject)

Bytecodeobject is an object in Quark-Engine that stores information about a single bytecode command.



After bytecode generated by `androguard`, it will look like below.

mnemonic	registers	parameter
invoke-virtual	v3	Lcom/google/progress/APNOperator;->deleteAPN()Z

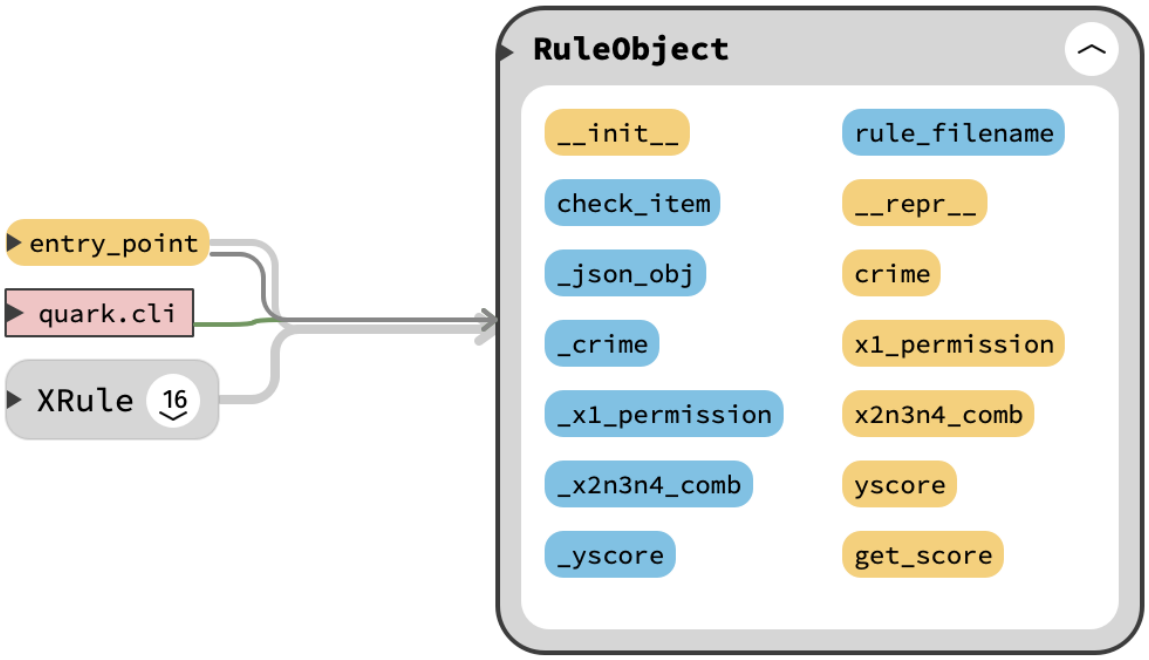
mnemonic: mnemonic is a human-readable instruction form compared to the machine code.

registers: registers used by each line of instructions in the smali bytecode instruction. They usually express in the form of "v3", "v4".

parameter: is the function or parameter used in the smali bytecode instruction.

10.3.5 RuleObject(quark.Objects.ruleobject)

The ruleobject in Quark-Engine will read a `JSON` format rule from a given JSON file, for example, sendLocation_SMS.

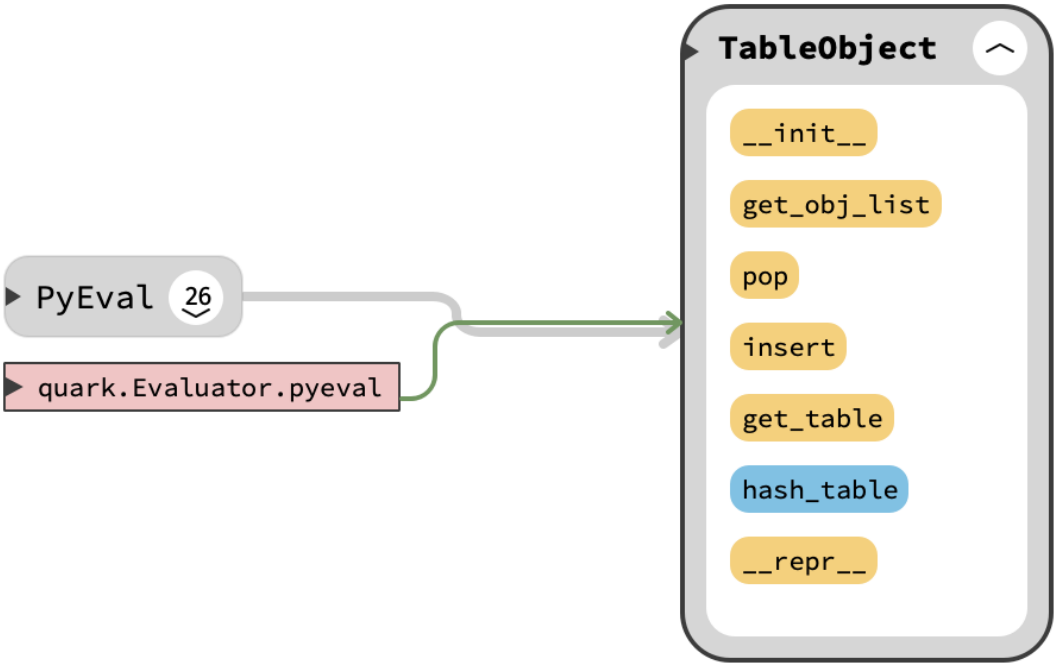


Explanation of each function

- **get_score**:
  - Returns the weight score based on the five stages of malicious behavior.

10.3.6 TableObject(quark.Objects.tableobject)

The tableobject is used to track the usage of variables in the register. We want to know whether the same variable is used by the two APIs we have defined in our rule.



In the output of the bytecode instruction generated by `androguard`, it is difficult to use a single register to track the usage of the same parameter being APIs call since the register is often reused by the Dalvik machine. Therefore, we creat the TableObject to store each RegisterObject in each row.

Tableobject is composed of multiple registerobjects as below:

[[registerobject,registerobject,registerobject],[registerobject,registerobject],[registerobject],[],[]]

Take this bytecode instruction as an example:

We can know what the v2 register stores, which is used by the send function.

However, the "v2" register may be overwritten by other values in the future. As above instruction, the content in v2 becomes "gps" after executing `const-string v2 'gps'`.

Therefore, the method we want to track must be reversed. We use the function call name as the tracker, and record the contents of the current register in the table.

Once we encounter a function call, we will check the parameters used by it, such as the above command invoke-virtual, then find out the current value of the v2 register, together with the function recorded in the table, assuming v2 finds the value "hello", the function name and parameter, "hello", will be recorded in called_by_func column.

You can take a look to this Bytecode example. With this table record, we can track all parameter content by each function call.

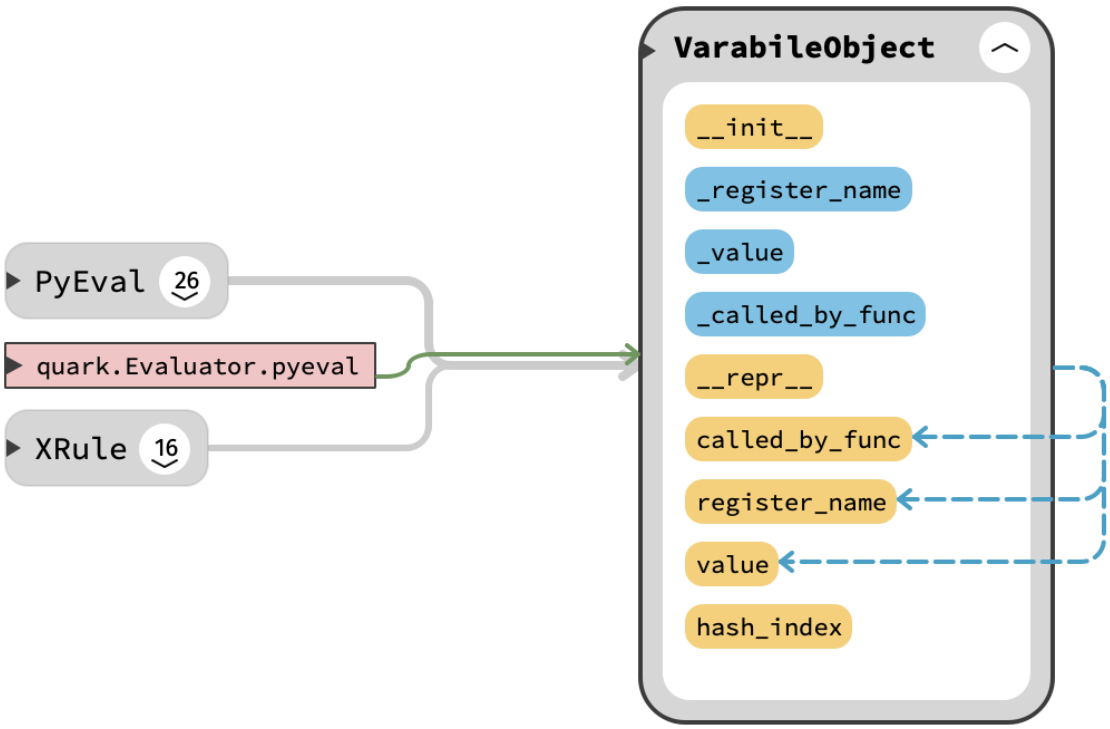
Explanation of each function

- **insert**:
  - Insert RegisterObject into the nested list in the hashtable.
- **get_obj_list**:
  - Return the list which contains the RegisterObject.

- **get_table**:
  - Get the entire hash table.
- **pop**:
  - Override the built-in pop function, to get the top element, which is RegisterObject on the stack while not delete it.

10.3.7 RegisterObject(quark.Objects.registerobject)

RegisterObject is used to record the state of each register. Each initialized registerobject will have register_name, value, called_by_func in a single instance.



register_name	value	called_by_func
“v3”	“GPS”	Lcom/google/progress/APNOperator;->deleteAPN()Z

register_name: register name, such as “v3”, “v4”.

value: the value stored in the register.

called_by_func: what functions are called with this register as a parameter.

Explanation of each function

- **hash_index**:

- Get the index number from given VarabileObject, given “v34” will return 34.

CHAPTER 11

quark

11.1 quark package

11.1.1 Subpackages

quark.core package

Subpackages

quark.core.axmlreader package

Module contents

quark.core.interface package

Submodules

quark.core.interface.baseapkinfo module

class quark.core.interface.baseapkinfo.**BaseApkinfo**(*apk_filepath: Union[str, os.PathLike], core_library: str = 'None'*)

Bases: object

activities
Return all activity from given APK.

Returns a list of all activities

all_methods
Return all methods including Android native API and custom methods from given APK.

Returns a set of all method MethodObject

android_apis
Return all Android native APIs from given APK.

Returns a set of all Android native APIs MethodObject

apk_filename

apk_filepath

core_library

custom_methods
Return all custom methods from given APK.

Returns a set of all custom methods MethodObject

filename
Return the filename of apk.

Returns a string of apk filename

filesize
Return the file size of apk file by bytes.

Returns a number of size bytes

find_method(*class_name: Optional[str] = '.*'*, *method_name: Optional[str] = '.*'*, *descriptor: Optional[str] = '.*'*) → quark.core.struct.methodobject.MethodObject
Find method from given class_name, method_name and the descriptor. default is find all method.

Parameters

- **class_name** – the class name of the Android API
- **method_name** – the method name of the Android API
- **descriptor** – the descriptor of the Android API

Returns a generator of MethodObject

get_method_bytecode(*method_object: quark.core.struct.methodobject.MethodObject*) → Set[quark.core.struct.methodobject.MethodObject]
Return the corresponding bytecode according to the given class name and method name.

Parameters **method_object** – the MethodObject instance

Returns a generator of all bytecode instructions

get_strings() → str

get_subclasses

get_wrapper_smali(*parent_method: quark.core.struct.methodobject.MethodObject*, *first_method: quark.core.struct.methodobject.MethodObject*, *second_method: quark.core.struct.methodobject.MethodObject*) → Dict[str, Union[quark.core.struct.bytecodeobject.BytecodeObject, str]]
Return the dict of two method smali code from given MethodObject instance, only for self-defined method. :param method_analysis: :return:

{ “first”: “invoke-virtual v5, Lcom/google/progress/Locate;->getLocation()Ljava/lang/String;”, “second”: “invoke-virtual v3, v0, v4, Lcom/google/progress/SMSHelper;->sendSms(Ljava/lang/String; Ljava/lang/String;)I” }

lowerfunc(*method_object: quark.core.struct.methodobject.MethodObject*) → Set[quark.core.struct.methodobject.MethodObject]
Return the xref from method from given MethodObject instance.

Parameters **method_object** – the MethodObject instance

Returns a set of all xref from functions

md5
Return the md5 checksum of the apk file.

Returns a string of md5 checksum of the apk file

permissions
Return all permissions from given APK.

Returns a list of all permissions

ret_type

superclass_relationships

upperfunc(*method_object: quark.core.struct.methodobject.MethodObject*) → Set[quark.core.struct.methodobject.MethodObject]
Return the xref from method from given MethodObject instance.

Parameters **method_object** – the MethodObject instance

Returns a set of all xref from functions

Module contents

quark.core.struct package

Submodules

quark.core.struct.bytecodeobject module

class quark.core.struct.bytecodeobject.**BytecodeObject**(*mnemonic*, *registers*, *parameter*)
Bases: object

BytecodeObject is used to store the instructions in smali, including mnemonic, registers, parameter

mnemonic
Dalvik bytecode instructions set, for example 'invoke-virtual'.

Returns a string of mnemonic

parameter
Commonly used for functions called by invoke-kind instructions, for example 'Lcom/google/progress/APNOperator;->deleteAPN()Z'.

Returns a string of the function name

registers
Registers used in Dalvik instructions, for example '[v3]'.

Returns a list containing all the registers used

quark.core.struct.methodobject module

class quark.core.struct.methodobject.**MethodObject**(*class_name: str*, *name: str*, *descriptor: str*, *access_flags: str = ''*, *cache: object = None*)

Bases: object

Information about a method in a dex file.

access_flags = ''

cache = None

full_name

is_android_api() → bool

quark.core.struct.registerobject module

class quark.core.struct.registerobject.**RegisterObject**(*register_name*, *value*, *called_by_func=None*, *value_type=None*)

Bases: object

The RegisterObject is used to record the state of each register

called_by_func
Record which functions have been called by using this register as a parameter.

Returns a list containing function name

current_type
Get the type of the value in the register

Returns a plant text that describes a data type

Return type str

hash_index
Get the index number from given VarabileObject.

Returns an integer corresponding to the register index

register_name
Individual register name, for example 'v3'.

Returns a string of register name

type_histroy

value
The current value stored in the register.

Returns a string of the value

quark.core.struct.ruleobject module

class quark.core.struct.ruleobject.**RuleObject**(*ruleJson: os.PathLike*, *jsonData: dict = None*)

Bases: object

RuleObject is used to store the rule from json file

api
Key native APIs that do the action and target in order.

Returns a list recording the APIs class_name and method_name in order

check_item

crime
Description of given crime.

Returns a string of the crime

get_score(*confidence*)
According to the state of the five stages, we calculate the weighted score based on exponential growth. For example, we captured the third stage in five stages, then the weighted score would be (2^3-1) / 2^4.

2^(confidence - 1)

Parameters **confidence** –

Returns floating point

label
A list contains various lebels described in https://github.com/quark-engine/quark-rules/blob/master/label_desc.csv

Returns a label list defined in rules

permission
Permission requested by the apk to practice the crime.

Returns a list of given permissions

rule_filename

score
The value used to calculate the weighted score

Returns integer

quark.core.struct.tableobject module

class quark.core.struct.tableobject.**TableObject**(*count_reg*)
Bases: object

This table is used to track the usage of variables in the register

get_obj_list(*index*)
Return the list which contains the VariableObject.

Parameters **index** – the index to get the corresponding VariableObject

Returns a list containing VariableObject

get_table()
Get the entire hash table.

Returns a two-dimensional list

hash_table

insert(*index*, *var_obj*)
Insert VariableObject into the nested list in the hashtable.

Parameters

- **index** – the index to insert to the table
- **var_obj** – instance of VariableObject

Returns None

pop(*index*)
Override the built-in pop function, to get the top element, which is VariableObject on the stack while not delete it.

Parameters **index** – the index to get the corresponding VariableObject

Returns VariableObject

Module contents

Submodules

quark.core.analysis module

quark.core.apkinfo module

quark.core.parallelquark module

quark.core.quark module

quark.core.rzapkinfo module

Module contents

quark.evaluator package

Submodules

quark.evaluator.pyeval module

class quark.evaluator.pyeval.**PyEval**(*apkinfo*)
Bases: object

AGET_KIND(**kwargs*)

AGET_WIDE_KIND(**kwargs*)

APUT_KIND(**kwargs*)

APUT_WIDE_KIND(**kwargs*)

BINOP_KIND(**kwargs*)

CAST_TYPE(**kwargs*)

CONST(**kwargs*)

CONST_CLASS(**kwargs*)

CONST_FOUR(**kwargs*)

CONST_HIGHSIXTEEN(***kwargs*)

CONST_SIXTEEN(***kwargs*)

CONST_STRING(***kwargs*)

CONST_WIDE(***kwargs*)

CONST_WIDE_HIGHSIXTEEN(***kwargs*)

CONST_WIDE_SIXTEEN(***kwargs*)

CONST_WIDE_THIRTY_TWO(***kwargs*)

FILLED_NEW_ARRAY_KIND(***kwargs*)

INVOKE_CUSTOM(*instruction*)

INVOKE_DIRECT(***kwargs*)

INVOKE_INTERFACE(***kwargs*)

INVOKE_POLYMORPHIC(*instruction*)

INVOKE_STATIC(***kwargs*)

INVOKE_SUPER(***kwargs*)

INVOKE_VIRTUAL(***kwargs*)

INVOKE_VIRTUAL_RANGE(***kwargs*)

MOVE_KIND(***kwargs*)

MOVE_RESULT(***kwargs*)

MOVE_RESULT_OBJECT(***kwargs*)

MOVE_RESULT_WIDE(***kwargs*)

NEG_AND_NOT_KIND(***kwargs*)

NEW_ARRAY(***kwargs*)

NEW_INSTANCE(***kwargs*)

static get_method_pattern(*class_name: str*, *method_name: str*, *descriptor: str*) → str

Convert a method into a string representation to record method calls during the tainted analysis.

Parameters

- **class_name** – the class name of the method
- **method_name** – the name of the method
- **descriptor** – the descriptor of the method

Returns a string representation of the method

show_table()

quark.evaluator.pyeval.**logger**(*func*)

Module contents

quark.forensic package

Submodules

quark.forensic.forensic module

quark.forensic.vt_analysis module

Module contents

quark.script package

Subpackages

quark.script.frida package

Module contents

Submodules

quark.script.ciphey module

quark.script.objection module

Module contents

quark.utils package

Submodules

quark.utils.colors module

quark.utils.colors.**black**(*text*)

quark.utils.colors.**blue**(*text*)

quark.utils.colors.**bold**(*text*)

quark.utils.colors.**color**(*text*, *color_code*)
Colorize text. @param text: text. @param color_code: color. @return: colorized text.

quark.utils.colors.**colorful_report**(*arg0*)

quark.utils.colors.**cyan**(*text*)

quark.utils.colors.**green**(*text*)

quark.utils.colors.**lightblue**(*text*)

quark.utils.colors.**lightyellow**(*text*)

quark.utils.colors.**magenta**(*text*)

quark.utils.colors.**red**(*text*)

quark.utils.colors.**white**(*text*)

quark.utils.colors.**yellow**(*text*)

quark.utils.graph module

quark.utils.output module

quark.utils.pprint module

quark.utils.regex module

quark.utils.regex.**extract_content**(*string*)

quark.utils.regex.**extract_file**(*string*)

quark.utils.regex.**extract_ip**(*string*)

quark.utils.regex.**extract_url**(*string*)

quark.utils.regex.**validate_base64**(*sb*)

quark.utils.regex.**validate_ip_address**(*ip*)

quark.utils.regex.**validate_url**(*url*)

quark.utils.tools module

quark.utils.tools.**contains**(*subset_to_check*, *target_list*)
subset_to_check = [“getCellLocation”, “sendTextMessage”] target_list = [“put”, “getCellLocation”, “query”, “sendTextMessage”] then it will return true. ——————————————————————— subset_to_check = [“getCellLocation”, “sendTextMessage”] target_list = [“sendTextMessage”, “put”, “getCellLocation”, “query”] then it will return False.

quark.utils.tools.**descriptor_to_androguard_format**(*descriptor*)

quark.utils.tools.**filter_api_by_usage_count**(*data*, *api_pool*, *percentile_rank=0.2*)
Sorting APIs by the number of APIs used in APK, and split APIs into P_set (less used number) and S_set (more used number) by percentile_rank (default 20%).

Parameters

- **data** – the object of apkinfo.
- **api_pool** – the APIs list for rule generation.
- **percentile_rank** – the int for rank of percentile.

Return P_set a set of APIs that less used.

Return S_set a set of APIs that more used.

quark.utils.tools.**get_arguments_from_argument_str**(*argument_str: str*, *descriptor: str*) → List[Any]
Get arguments from an argument string.

Parameters

- **argument_str** – string that holds multiple arguments and uses commas as separators

- **descriptor** – string that holds a descriptor for type inference

Returns python list that holds the arguments

quark.utils.tools.**get_parenthetic_contents**(*string: str*, *start_index: int*) → str
Get the content between a pair of parentheses.

Parameters

- **string** – string to be parsed
- **start_index** – index to specify the parenthesis

Returns string holding the content

quark.utils.tools.**remove_dup_list**(*element*)
Remove the duplicate elements in given list.

quark.utils.weight module

class quark.utils.weight.**LEVEL_INFO**
Bases: enum.Enum

An enumeration.

High = 'High Risk'

LOW = 'Low Risk'

Moderate = 'Moderate Risk'

class quark.utils.weight.**Weight**(*score_sum*, *weight_sum*)
Bases: object

calculate()

Module contents

quark.webreport package

Submodules

quark.webreport.generate module

class quark.webreport.generate.**ReportGenerator**(*json_report*)
Bases: object

This module is for web report generating.

get_analysis_report_html()
Load the quark JSON report and generate the HTML of the Quark web report.

Returns the string of Quark web report HTML

get_rule_generate_editor_html()
Load the rule generation result and generate the HTML of the Quark web report.

Returns the string of Quark web report HTML

insert_genrule_report_html(*data*, *filename*, *md5*, *filesize*, *rule_number*)
Generate the HTML of rule generation result secton.

Parameters **data** – the dict of rule generation result

insert_radarechart_html(*five_stages_labels*, *all_labels*)
Generate the HTML of radare chart secton in Quark web report.

Parameters **five_stages_labels** – the set of lebels

with 100% confidence crimes :param all_labels: the set contain all labels with crimes above 0% confidence

insert_report_html(*data*)
Generate the HTML of summary report secton in Quark web report.

Parameters **data** – the dict of Quark JSON report

insert_sample_information_html(*rules_number_set*, *filename*, *md5*, *filesize*, *labels*)
Generate the HTML of sample information secton in Quark web report.

Parameters **rules_number_set** – the dict of rule number

for each confidence :param filename: the string of the sample filename :param md5: the string of the sample md5 hash value :param filesize: the string of the sample filesize :param labels: the set of lebels with 100% confidence crimes

quark.webreport.generate.**count_confidence_rule_number**(*data*, *confidence*)
Get the number of rules with given confidence in JSON report.

Parameters

- **data** – the dict of Quark JSON report
- **confidence** – the string of given confidence

Returns the int for the number of rules

with given confidence in JSON report

quark.webreport.generate.**get_all_labels**(*data*)
Get all labels with crimes above 0% confidence.

Parameters **data** – the dict of Quark JSON report

Returns the set contain all labels with crimes above 0% confidence

quark.webreport.generate.**get_five_stages_labels**(*data*)
Get the labels with 100% confidence crimes.

Parameters **data** – the dict of Quark JSON report

Returns the set contain all lebels with 100% confidence crimes

quark.webreport.generate.**get_json_report_html**(*layout*, *data*)
Convert the quark JSON report to HTML with script tag.

Parameters **data** – the dict of Quark JSON report

Returns the string of Quark JSON report HTML

Module contents

11.1.2 Submodules

11.1.3 quark.cli module

11.1.4 quark.config module

11.1.5 quark.freshquark module

11.1.6 quark.logo module

quark.logo.**logo**()
Output our amazing logo

Returns None

11.1.7 quark.radiocontrast module

11.1.8 quark.report module

11.1.9 quark.rulegeneration module

11.1.10 Module contents

CHAPTER 12

FAQ

12.1 I have some questions. Where can I ask?

We welcome you to post your questions on our GitHub repository or our Telegram. We'll try our best to answer your questions as soon as possible, but not be instant since we're focusing on adding new features into Quark-Engine! Also, please understand that we may not answer questions about personal information since we want to keep them as our privacy.

12.2 I got an error while using Quark-Engine. What can I do?

Please use `pip3 install quark-engine --upgrade` to update your Quark-Engine and use `freshquark` to update rules first, then inspect if there are misspellings in the command. Here are two common errors that will occur. If those errors happened to you too, please check out the following information.

1. Errors on arguments (Reference to issue #239): This type of error is usually caused by an outdated version of packages. Please update Quark Engine and the related python package first, then check if the problem still exists.
2. Errors on rules not found (Reference to issue #237): Please update Quark-Rule with `freshquark` first. Since the way to specify rules is by adding `<path_to_the_rule>` as an argument, you need to input `<path_to_the_rule>` if the rule file is not in the current folder. If you want to select one of the rules of Quark-Rule, the default path to Quark-Rule is `$HOME/.quark-engine/quark-rules/`.

12.3 How do threshold, score, and weight working in Quark Engine?

About those details, we have a video to explain how it works. You can check out the video on YouTube: https://www.youtube.com/watch?v=SOH4eqrv9_g

12.4 Why do scores keeping the same in all the analyses?

The default value is one since we would like users to define these numbers themselves, and we are still doing experiments to adjust the numbers.

12.5 How can I write a rule?

We have a detailed introduction to add rules. You can check it out here.

12.6 How can I contribute my rules?

Feel free to make a pull request on the Quark-Rule repository. We appreciate your contribution to Quark-Engine!

12.7 Can I take part and contribute to Quark?

That's a big YES! We welcome anyone interested of Quark-Engine. Please check out our development document and join our Telegram.

CHAPTER 13

Indices and tables

- genindex
- modindex
- search

Python Module Index

q

Index

T

U

V

W

Y