國立政治大學資訊管理學研究所

碩士學位論文

AppScan:手機應用程式行為靜態偵測掃描-以iOS 為例

AppScan : Static mobile application behavior scanning on iOS executable

指導教授：郁 方 博士

研究生：王 韋 仁 撰

中 華 民 國 一〇六 年 七月

Abstract

Mobile application is the most popular and dominant software applications nowadays, so the actual behaviors of the application and the related security and privacy issues become more and more important. On the other hand, as time goes by, there are more and more applications on the AppStore stop to update or being abandoned but not removed from AppStore. However, the users know nothing about the lack of maintenance problems and still download and use it. In this research, we will resolve the issue for checking specific property method sequence within an application. By using IDApro[29] to generate function call graph and the subroutine control flow graphs, we use syntax checking strategy to perform a across subroutines sequential checking solution. We will check the application behavior by predefining a property method sequence as pattern and then check with applications'. The analysis method can illustrate whether a property method sequence exists in the application which is available on App Store. This may help us to check some malicious behavior property method sequence or specific behavior method sequence (ex. using deprecated api methods) in the applications. We have prepared some property method sequence as our system input pattern extracted from all the available iOS SDK methods fetching by our web crawler. We will check whether an application contains the prepared method sequence or not. If the sequence exists in the application, we would record the method sequence call included in the subroutine within the application. Then the results data will be aggregated in our database, and export as api service for visualizing and statistic uses. Finally, we construct a call sequence analysis system for the above checking functions and show the result in a web service form.

Contents

List of Figures

1 Introduction

Mobile applications (To be short, I will call it "apps" in the following paragraphs) become popular in recent years due to the rise of smart devices and application stores. On Apple WWDC 2015 (World Wide Developer Conference 2015) [22] , Timothy Donald Cook, the CEO of Apple.Inc, announced that the download times of apps has reached 100 billion on Apple's official app store during the 7 years. Besides, Cook also claimed that there are 1.5 million apps in the App Store, and still increasing rapidly. The tremendous amount implies the fact that mobile apps are extremely close to our daily lives. In the app store, we could simply find almost any type of apps we need. With the assistant of these apps, we can enjoy convenient communication, better productivity, various music and games, thus simplifying our works and enriching our lives. However, though we frequently use applications in our daily life, we do not completely realize how an application do when the system is executing. Also, we can not confirm the claim of an application that they are not offensive to our personal and confidential data is true or not. A study ProtectMyPrivacy: Detecting and Mitigating Privacy Leaks on iOS Devices Using Crowdsourcing [7] pointed that most of applications were developed by certain companies with scale and trusty, and more small companies and developers develop mobile apps in recent years, and these developers may access private data at will, and these privacy data may leak from mobile device and being abuse. Further, there are many malicious applications around us, especially in applications downloading from non-official repositories, such as Cyndia. However, relatively large proportion of users would jailbreak their iDevices in order to download applications from the non-official repositories without any defensive measures on their device. Many security issues have to be concerned in current mobile dominant software applications. Therefore, in this research, we perform a system, AppScan, to clarify an application behavior by their program context, but not by the words and descriptions provided by application developers. In this way, we will be more confident to know what an application actually do in our mobile system. As we have mentioned previously, the objective of this research is to clarify what an application actually do in our system, and we would like to further analyze how most developers implement an application behavior, what classes or methods they use and if there's any pattern existing or not. Therefore, we make AppScan can continuously and automatically fetch new application from App store, and quickly judge that if the specific behavior we want to check is existed in the application or not.

For well implement our system in this research, we took a survey of works related to mobile applications behaviors and how to detect these behaviors with reverse engineering. Besides, we will encounter problems when processing lots of tasks in parallel computing, so we will also study the issues about distributed computing.

1.0.1 Strategy of sandbox policy and iOS program standard agreement

Apple Inc. has a strategy for system stability, and compulsively lock some system features in the official developer standard agreement - iOS Developer Program License Agreement[6], and the sandbox policy is to achieve the target. It will only permit and dispatch part of the resources in system for each executing application. Therefore, developers or users are forbidden to get the root right inside the iOS system. To some degree, the strategy keeps the system's performance and stability. However, it limits some users' rights when manipulating in iOS.

1.0.2 Public, private frameworks and private APIs on iOS

The officially provided iOS frameworks can be divided into two forms, traditional C functions exported by shared library and methods in Objective-C classes that are managed at the runtime. However, only public frameworks can be used by the third party developers because the private frameworks have private APIs [30], and even some public frameworks may contain private APIs. Both of the private APIs are forbidden to use by third-party developers or it may cause security and user privacy offensive issues[26] in iOS system.

1.0.3 Deprecated APIs

There will have some methods outdate in iOS SDK, with the version updated. These methods name as "Deprecated APIs". The Deprecated APIs are the methods or classes that are outdated and will eventually be removed by the Apple Inc.. Generally speaking, Apple Inc. will deprecate API when they introduce a replacement, usually because the new one can take advantage of new hardware, operating system or programing language feature.

For example, from iOS8 Apple Inc. provide new UIAlertController class which you can use instead of UIAlertView which is deprecated, it is also stated in depreciation message "UIAlertView is deprecated. Use UIAlertController with a preferredStyle of UIAlertControllerStyleAlert instead".

These old methods will not terminate immediately, and the Apple Inc. will still support these methods for a while. When developers use some method and the deployment target is set to equal or a higher version when the method was deprecated, they will get the compiling warnings. Once the Apple Inc. decided to remove some deprecated APIs, and those applications use the deprecated APIs will not be supported anymore, and may lead to crush problem in these applications.

Eventually, these deprecated apis will be terminated in SDK, so developers should follow the system update information, and avoid using the deprecated apis.

A massive applications context analysis need much computing powers, it is necessary to enforce our efficiency by using distributed computing solutions. To solve the problems and speed up the system performance, we adopt MapReduce[11], a distributed computing model, and implement the model idea with Hadoop[1], the Apache open-source software project, for the target of providing a reliable, scalable, distributed computing solution.

We take Hadoop Definitive Guide[34] as our reference and build up a hadoop cluster with general hardware to offer enough computing power and scalable storage space with hadoop file system(HDFS) to achieve our distributed infrastructure environment. Then, we implement our sequence analysis algorithm with MapReduce model, and adopt many useful features in our map function thus reducing functions in the MapReduce Design Patterns[14].

1.0.4 Hadoop

Hadoop[1] is an apache project provides a solution that we can build a reliable distributed environment with some commodity hardware. By linking these hardware, it can provide more computing power, memory and hard-disk storage. Moreover, these resources own high tolerance and good scalability.

1.0.5 MapReduce

MapReduce[12] was proposed by Dean and Ghemawat of Google Inc. The basic idea of this distributed computing model is the divide and conquer concept. We will construct one or many map functions for digesting repeat tasks in our job, and reducing functions to gather the results from map functions. After processing literately, we will converge a final result in our job. Through the divide and conquer algorithm, we can figure out huge data processing problem.

2 Literature review

In recent years, more and more studies concerned about mobile applications' behavior, related privacy and security issues. Early in the "A survey of mobile malware in the wild"[17], it discusses many isssues about leaking of the user sensitive information, and later in year 2013, a survey on security for mobile devices[23], they describe many different types of mobile malware and predictable potential issue in many different mobile apps operating system. The behviors of apps were not only disscuss in any single platform, it is a general issues in both iOS and android operating system. Because of the different official policy between Google Inc. and Apple Inc. on apps. Android apps are more open than apps on iOS apps. Therefore, many statements claim that the iOS apps are more safer than android apps. However, in the research "Android or iOS for better privacy protection?"[20] are also clearly analysis the privacy and security issues on both two platforms, and further points that additional SS-APIs(Security sensitive APIs) are always invoked on iOS, and these may cause higher risks on privacy leak. In these researchs, they have developed many analysis tools and methods. However, the bolcked policy on iOS apps lead the different research progress between android system and iOS system. In contrast of iOS platform, most of these studies focus on android platform, and many analysis tools were built for android apps only such as TaintDroid[16], AsDroid[21], FlowDroid [8], Droidra[25] and the research "Static analysis of implicit control flow: resolving Java reflection and Android intents"[31].

In general, two different approches are use in these research to detect behaviors of apps, the static analysis and the dynamic analysis.

Static analysis methods are usually use on application behavior detectations. In Flowdroid[8], it adopts static analysis by building a control flow graph of API methods call within android app for detecting privacy leakage. However, the dynamic loaded classes problems will be ignored by only construct the control flow graph. Futher, in Droidra[25] they use constant propagation solver to solve this problem, and the "Static analysis of implicit control flow: resolving Java reflection and Android intents"[31] is also facing the problem, they focus on control flow related to Java reflection and android intent checking of applications and they also use static analysis on constructing control flow graph, and in the Appintent[35], and IccTA[24], they both adopt static taint analysis to preprocess and extract the information from apps. Appintent focus on all possible data transmission paths and possible events related to each path.

However, static analysis have the limitations on checking the behavior that trigger by external input in application runtime, and the dynamic checking related methods can be apply with this situation. Dynamic analysis will observe application by actually executing it. The TaintDroid[16] is one of researches that adopt dynamic anaylsis. They detect privacy-sensitive data and tracing data will be passed to external or not as a risk evaluation and perform a dynamic taint analysis tool to detect sensitive data leaves from device, and for another research, IccTA detects specific ICC (Inter-Component Communication, a base communication model provided by android os) links and leak in android apps.

Apparently, there are relatively rare studies on iOS than android. PiOS[15] is an important study for detecting privacy leak in iOS apps, they introduce a way to reconstruct apps CFG (Control flow graph) from binaries of apps, and checking the privacy leak with data flow.

In PSiOS[33], and Jekyll on iOS[32], they are not only detect the leak of privacy data, they also provide a tool to prevent the leak, and the AppBeach[36] uses the static way to check the possible malicious behavior pattern in an application. Then, iRiS[13] reveals issues of abusing the private API illegally in iOS apps with both static and dynamic analysis.

Besides, in these researches, they also use many other aspects to analysis mobile apps, and eager to solve these malware application issues. In the PMP[7], they gather the feedbacks from real users, and build up this system to receive and analysis the applications' behavior in crowdsourcing power. In Checking app behavior against app description[19], they clustered apps with their description topic, and identify the sensitive APIs usage in each cluster, and the Apposcopy[18] adopt static analysis along with semantics approach to figure and detect the malware apps issue. The AsDroid[21] focus on the UIUX logic conflict with the program behavior and regard these abnormal situation as strange and need to be removed. The research "Hey, you, get off of my market: detecting malicious apps in official and alternative android markets."[38] detect the apps accroding the behaviors that need to request permission with user in app.

In this research, we are going to check iOS applications by adopting static analysis and sequential checking method with distributed computing framework, Hadoop, and we want to study the deprecated APIs of iOS SDK usage and specific API usage pattern within these applications. The deprecated APIs issues are very common problems in software, and many studies are related to this, they explore many different points like APIs retrospective[37] and design or programing issues of empirical[27] [10]. In this paper, we will identify the deprecated usage and many defined patterns of iOS applications. It can help to improve the bad or wrong using APIs problems in iOS applications.

3 Application properties and checking

Before we get start to process the applications, it is necessary to define some behavior properties may be found in an app. In our research, we defined a sequence of method calls as a application behavior property. We will prepare a behavior property filter list for those behaviors we are interested in, then we will use it to check by our distributed sequencial analysis program. These property filters are consists of several aspects. The first one we focus on the methods that are deprecated by iOS SDK. These methods may create the risk of crash issues when operating system updated by Apple Inc., and the problems will cause the inconvenient for users who downloaded the apps. The second part is related to some specific method usages which is especially mentioned in iOS API developer reference[2] such as asking for user permissions, or asking for device locations..., the Apple Inc. given a sample to demonstrate how to use these methods to perform the property feature, so we called these properties as "Golden rules".

On the order hand, in order to check the correct context in binary, we have to decrypt apps downloaded from App Store in advance and convert them into readable assembly files with annotations. By using disassemble tool, IDA pro[29], we can turn decrypted binary into assembly file. Every application assembly file contains many subroutine blocks, and when we dive into each subroutine we will get more small logic area called locations that can help us analyze the code and realize the classes or methods invoked in the context. We check context subsequence in assembly files by behavior properties, and use distributed computing to do the sequence analysis. Before analyzing, we have to do some data processing and define the meaning of subsequence in our experiment. The details are described below.

3.1 iOS developer api reference web crawler

In order to get the frameworks, classes, and methods in iOS SDK, we have to fetch the data from the iOS developer API reference site[2]. However, the web pages are made by HTML and CSS contents, so we need to extract the information we need then format and store them in our database. Therefore, we develop a web crawler to process the data we need.

Our crawler are basically designed in three parts as show in Figure 1, including the services, parsers, and stores. The services are reusable, mostly they are used to handle with file write out and initial http requests. The parsers are used to parse html content and select the data we need, so basically a parser will map to one or more target web pages. Stores can be regarded as the data model, we will define the model to shape our data object and implement the serialize methods.

Therefore, we will dispatch http request to the target page in the iOS developer API reference site[2], and fetch the data, parsing with our parsers, collect and format the data into the object we defined in stores. It's a pipeline working flow to send request, process, format, collect and write out. Finally, when we got the output files (we use the json format as output), we will further to write the json into our database.

Through this process flow, we can fetch the API data from the Apple official site, and generate the deprecated api list and method sequence patterns we need.

3.2 Definition of an application property

In this part we will define the meaning of a behavior property in our analysis, and explain how we generate a behavior property by method sequence calls as our system input.

3.2.1 Behaviors in iOS applications

iOS executables usually have to apply many methods to complete some specific purpose such as connecting to bluetooth, sending SMS messages...etc. Most of these behaviors are implemented by the methods provided by official iOS SDK. In other words, most of

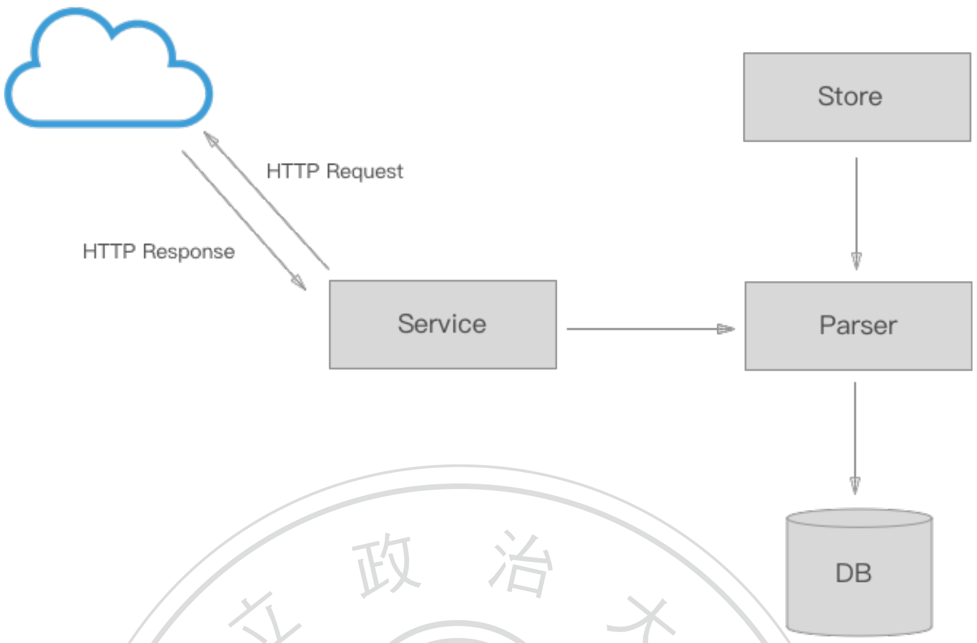


Figure 1: The web crawler structure of iOS developer reference guide.

applications can accomplish their purpose by using the SDK provided methods.

Fortunately, the framework naming rules are always related to the functionalities, for example, MapKit framework will contain many classes and methods about usages of map funcitons in iOS SDK. Thus, it is much more convenient for us to clearify the usages of these classes and methods. Because the SDK defines many specific behaviors in official documents, we use these behaviors as targets of our application sequence analysis.

By developing our web crawler, we can easily fetch and sort out the frameworks, classes and methods information from the iOS developer API reference[2] web site. Then we turn these methods into many lists classified by their different behaviors, and take these list as a standard to check the behaviors of our applications downloaded from App store.

3.2.2 iOS API reference best practice

When we build up an iOS application, we must use lots of methods in SDK (Software Development Kit) provided by Apple Inc. In most scenario we can directly use these methods with our business logic without authorized or permission required, however, in some cases (user privacy or mobile device native function related) we have to ask for permission when call the method. In Apple.Inc developer guide, it provide some example codes in a proper or suggested way for calling these SDK methods, and it also suggest that developers should follow the guideline to call these methods. According to above, in this research, we adopt the guidelines as a review standard, and use it to check available apps on app store are truly follow these guidelines or not.

3.2.3 Application property subsequence

We regard each method we may use in an application as a basic unit, and a series of methods in a specific sequence will form an application behavior that we called a property of an application, and a property always consists of developers' self-defined methods and official iOS SDK methods. Because we can't predict a developer how to name their self-defined methods, we will focus on analyze the methods provided by the iOS SDK, and skip the self-defined ones. Then, we can use some specific method sequence calls to generate a behavior property as our checking pattern, and check if there is any application property containing this pattern as its subsequence, thus representing the existence of a property in an app. For example, there may be a sequence of methods formed like "A-B-C-A-D-F-A" in an app, and we may define a pattern like "A-C-F-A". The former is our application property sequence extracted from source code of an application, and the latter is a behavior pattern we want to check with. Apparently, the sequence "A-C-F-A" is one of the subsequences in "A-B-C-A-D-F-A", so we can find that the pattern appears in this app, in other words, we can also say the application have this property.

3.2.4 Producing a property checking pattern

After defining the application property subsequence, we have to prepare method list that available in iOS SDK for us to select and compose an application property checking pattern. In order to do this, we extracted all the information we need from iOS Developer Reference by web crawler mentioned before, and unified all the lists of frameworks, and the subordinating classes and methods available in iOS SDK, and also, we format these information and convert into a more serializable structure. Therefore, we can start to generate our pattern in specific method call steps. For example, in objective-C, when we want to get a user location permission, we have to use some methods provided by class CLLocationManager in a proper way like the sample code in Listing 1

Listing 1: Get user location permission code snippet

```
- (void)requestAlwaysAuthorization
{
    CLAuthorizationStatus status = [CLLocationManager authorizationStatus];

    if (status == kCLAuthorizationStatusAuthorizedWhenInUse || status ==
       kCLAuthorizationStatusDenied) {
        NSString *title;
        title = (status == kCLAuthorizationStatusDenied) ? @"
            LocationServicesAreOff" : @"Background␣locationIsNoEnabled";
        NSString *message = @"....";

        UIAlertView *alertView = [[UIAlertView alloc] initWithTitle:title
            message:message delegate:self cancelButtonTitle:@"Cancel"
            otherButtonTitles:@"Settings", nil];

        [alertView show];
    } else if (status == kCLAuthorizationStatusNotDetermined) {
        [self.locationManager requestAlwaysAuthorization];
    }
}
```

From the codes in Listing 1, we can pick the method sequences as a checking pattern like:

1. CLLocationManager:authorizationStatus

2. CLLocationManager:requestAlwaysAuthorization

Next, we can take any application with the pattern as system input to compare. As the result, we can figure out an application with get user location permission behavior is follow the pattern or not.

3.3 Behavior pattern syntax checking

We adopt Hadoop as our distributed computing framework in our system to execute the checking jobs. However, we need to use IDA pro, a disassemble tool, to extract the application, then we can get start to execute our comparing.

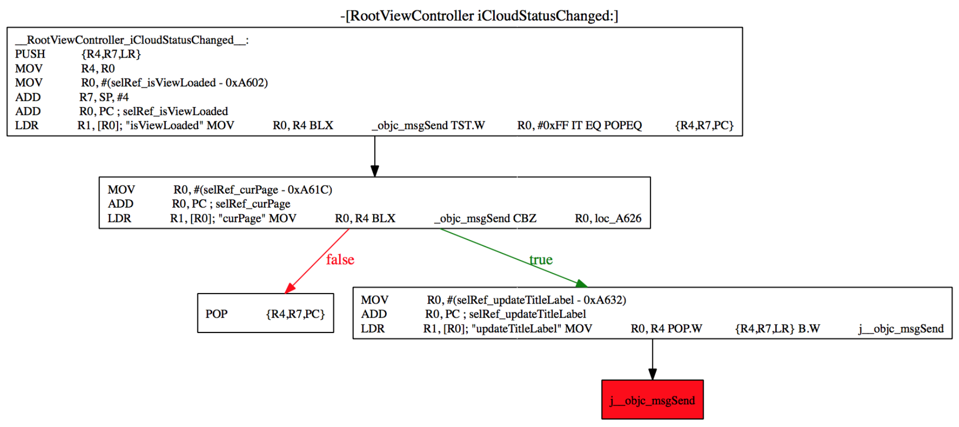
3.3.1 Automatically generated asm file, functions call dependency and subroutine function call dependency by IDA pro

IDApro provided a GUI (Graph User Interface) for users to execute disassemble jobs. However, we need to execute a batch of disassemble jobs. To figure the problem, we use a provided plugin idapython, helping us to use python script to manipulate the IDApro. Therefore, by using idapython, we can continuously execute our disassemble jobs.

3.3.2 Annotation generated by IDA pro

By using IDA pro to disassemble our iOS executable, we can get an .asm file of apps. In the same time, we can also dump the function calls graph Figure 3 in an app and the all subroutine call graphs Figure 2 within an app. These graphs construct by the GDL(Graph Description Language).

In the asm file, it will extract the class names and method names from the application, and put the names after lines of application assemble file as annotations with a prefix "selRef_" and "classRef_". With these information, we can conduct the sequential syntax checking. In the subroutine dependency graph, we can trace the method call sequence inside, and the functions call dependency graph display the dependency relation between every subroutine. With these information, we can start to execute the syntax checking and comparing.



```
__RootViewController_iCloudStatusChanged__:
PUSH        {R4,R7,LR}
MOV         R4, R0
MOV         R0, #(selRef_isViewLoaded - 0xA602)
ADD         R7, SP, #4
ADD         R0, PC ; selRef_isViewLoaded
LDR         R1, [R0]; "isViewLoaded" MOV        R0, R4 BLX        _objc_msgSend TST.W        R0, #0xFF IT EQ POPEQ        {R4,R7,PC}

MOV         R0, #(selRef_curPage - 0xA61C)
ADD         R0, PC ; selRef_curPage
LDR         R1, [R0]; "curPage" MOV        R0, R4 BLX        _objc_msgSend CBZ        R0, loc_A626

POP         {R4,R7,PC}

MOV         R0, #(selRef_updateTitleLabel - 0xA632)
ADD         R0, PC ; selRef_updateTitleLabel
LDR         R1, [R0]; "updateTitleLabel" MOV        R0, R4 POP.W        {R4,R7,LR} B.W        j__objc_msgSend
```

Figure 2: This picture show a graph of a subroutine within an application disassembled by IDA pro.

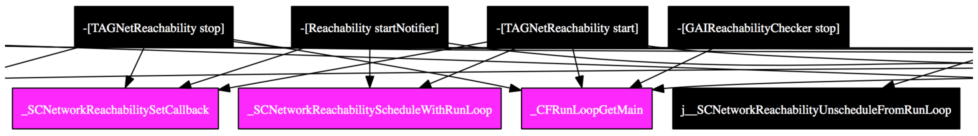

-[TAGNetReachability stop]
-[Reachability startNotifier]
-[TAGNetReachability start]
-[GAIReachabilityChecker stop]
_SCNetworkReachabilitySetCallback
_SCNetworkReachabilityScheduleWithRunLoop
_CFRunLoopGetMain
j__SCNetworkReachabilityUnscheduleFromRunLoop

Figure 3: This picture show a part of an application all functions (subroutines) calls dependency graph disassembled by IDA pro within an app.

3.3.3 Application property existence checking

After we get the annotations of method name and class name generated by IDA pro, we use them as key words to compare with the pattern we defined. First, we will check the whole application subroutines with their class names, and there will be a preliminary determination to check if the application has the behavior and decide that whether we need to dive into dig out the methods in it. Second, once we found a class name match with the class in the pattern, we will continuously check if the method names under this subroutine block match the methods in pattern. If so, we will say the application has the property. For example, we have the pattern like this:

1. CLLocationManager authorizationStatus
2. CLLocationManager requestAlwaysAuthorization

When the class ”classRef_CLLocationManager” is found in one subroutine block and matches the pattern class we gave, we will begin to check the subroutine methods and match with our pattern. In this example, we will start to check the methods "authorizationStatus" and "requestAlwaysAuthorization" is existed in the subroutine or not. Therefore, we can identify the usage of some specific methods within apps, in this research, we use this method to check the usage of iOS SDK deprecated APIs in apps.

3.3.4 Application property sequential checking with LCS in single subroutine

However, in most situations, a checking pattern is a sequential methods call, so we can't just find out the existence of method calls. We need to go further, checking the applications in sequence and compare the similarity with checking patterns.

Here we use LCS (Longest Common Subsequence) algorithm to execute the analysis, and we can find the most resembling sequence in the subsequence with checking patterns. For example, we may get a method call sequence from an application subroutine "A-B-D-A-E-F-C-C", and a user input method subsequence may be like "A-B-C-A-B-B". By using LCS, we can refer the result is "A-B-A", and the length of this result "3" can also be a index of similarity between checking pattern and a method call sequence.

Therefore, in our example, get user location permission in Listing 1, we may get a subroutine methods sequence from a application like this:

1. classRef_CLLocationManager
2. selRef_authorizationStatus
3. classRef_UIAlertView
4. selRef_alloc
5. classRef_UIAlertView
6. selRef_initWithTitle:message:delegate:cancelButtonTitle:otherButtonTitles:
7. classRef_CLLocationManager
8. selRef_requestAlwaysAuthorization

9. classRef_UIAlertView

10. selRef_show

Assume we adopt the same checking pattern defined in last paragraph, then we can execute the sequential checking. From this example, we can easily refer the result will be ["authorizationStatus","requestWhenInUseAuthorization"] and the similarity length is 2, and the result can represent that this application actually follow the checking pattern we defined.

Hadoop - FileInputFormat and RecordReader

We adopt Hadoop as our distributed computing solution, and store our data in the HDFS. Therefore, we have to implement our sequential algorithm under the Map-Reduce framework. The first step to process our assembly file is to read the file into our memory. In general condition, Hadoop provide a default FileInputFormat and RecordReader to solve this problem by reading file content line by line and set the key value pair as file offset and line content. However, in our condition, we have to make sure our line content sequence within each subroutine in a file, so we can't use default FileInputFormat and RecordReader to read our files. Instead that, we have to overwrite default FileInputFormat and RecordReader and build up our own implementation to make each subroutine as input content value and set the key with the assembly file subroutine name.

Algorithm 1 ExtractSbrt(asm)

```
Require: $asm$: The ARM assembly file of iOS APP
Ensure:
1: $r = false$;
2: $sbrt = ""$;
3: while $hasNextLine(asm)$ do
4:     $line = nextLine(asm)$;
5:     if $line \in REGEX(" \ =+SUBROUTINE=+")$ then
6:         $r = true$;
7:     end if
8:     if $r$ then
9:         $sbrt = concat(sbrt, line)$;
10:    end if
11:    if $line \in REGEX(";Endoffunction.*")$ then
12:        $writeKV(nameOf(F), sbrt)$;
13:        $sbrt = ""$;
14:        $r = false$;
15:    end if
16: end while
```

MapReduce - Map

In MapReduce framework, the mapper defined the job that will be distributed to execute. In our mapper, we will receive the subroutine form the RecordReader we modified and our behavior checking pattern. When we got these in map, we have to find that is there any class in pattern match with the subroutine class name call, so we have to process our subroutine by splitting the whole subroutine with empty space and maintain two list for store class name system calls and methods call sequence. Then we compare the class list with our pattern. if we got a match result, we will write out the method call sequence with class and method name as output key, appear times we gather in the subroutine as value.

Algorithm 2 Mapper($sbrt$)

```
Require: $sbrt$: The subroutine from a assembly file of iOS APP, $pattern$[]: Checking pattern list
Ensure:
1: $CLASS\_LIST = [];$
2: $METHOD\_LIST = [];$
3: while $hasNextLine(sbrt)$ do
4:     $line = nextLine(sbrt);$
5:     if $line \in REGEX(\text{"}classRef\_+\text{"})$ then
6:         Add found to $CLASS\_LIST;$
7:     end if
8:     if $line \in REGEX(\text{"}selfRef\_+\text{"})$ then
9:         Add fonded to $METHOD\_LIST;$
10:    end if
11: end while
12: if $interset(class \in pattern, CLASS\_LIST)$ then
13:     $LCS(METHOD\_LIST, method \in pattern)$
14:     $writeKV(name \in sbrt, metchedMethodResult)$
15: end if
```

MapReduce - Reduce

In reducer, we will get the result key-value fair form mapper. We have to further separately accumulate the appear times count for each key form different map output. When all the reduce jobs done, the result will be written out into HDFS. In our reducer, we are simple accumulate all the methods appear counts, then we will process the results, bundle all the method names and counting records to match our iOS SDK method indexes, add the related logs in database for statistics and data presenting.

3.3.5 Application property sequential checking in function call dependency graph and subroutine method dependency graph

In each subroutine method dependency graph, we can construct a directed graph data structure by using method call as the vertex and the call sequence represent in arrow directions. By go through these possible paths in each application subroutine graph with a limited path length, we can use the LCS method we mention before to check the application property.

Although a subroutine may call a iOS SDK method (external function) directly, it usually call the other subroutines and call iOS SDK method in that subroutine. If we directly check the application property with LCS in each subroutine graph respectively, it will lead many miss judge cause by ignoring the method call relations between subroutines. In other words, by using originally strategy, we can only find a property that just included in one subroutine. To improve this, we join the application function call dependency graph information to reveal the relations between subroutines. Although the function call graph only displays the sequential between subroutines, we improved original algorithm into a two stages LCS checking way, and this new checking method can figure out the property checking across many subroutines problem.

In first stage, we will compare each subroutine graph paths with our checking property in each subroutine, and we will get temporary result in this stage. Consider the cycle situation may happen in a graph, we will define a checking path length as limited. When we get a match LCS result under first limited length, we will increase length limited then checking again, until the result won't get more long LCS result than previous one. In here we will use the number of vertexes in a graph as limited length initially.

In second stage, we gather all results from first stage, we use these LCS results to represent the subroutines. With subroutine relations information in function call graph, we will use these LCS results to reconstruct this function call graph, then use LCS method to check the new graph with our checking property. Therefore, we can get the correct matching result, even the checking property across many subroutines.

To sum up the above-mentioned, we have to deal with three main things "Reconstruct Control Flow Graph", " Reconstruct function call graph" and "Compare Graph with pattern in LCS" to compose our analysis. To be more clear, we will go through our analysis detail below using our algorithms, and here is a whole checking processing example show in Figure 4. We use IDApro to generate the FCG(functoin call dependency graph), and all the control flow graphs of all subroutines. These graphs contain many information, included the method calls inside that we needed to use in the algorithm. However, these graphs can't not apply in our algorithm directly, we have to reformat and construct new graphs for our needed. Here in algorithm3 and algorithm4 show how we extrated the method calls and reformat the edges from the original graph and rebuild a new one by using method calls as vertex.

Algorithm 3 ReconstructureCFG(G_{raw})

```
Require: $G_{raw}$: The control flow graph(.graphml) of a subroutine within an iOS app.
Ensure:
 1: $G := \{\}$;                                   ▷ Empty result graph
 2: $M_{head} := \{\}$     ▷ Empty map which map vertex in $G_{raw}$ to vertex in $G$
 3: $M_{tail} := \{\}$     ▷ Empty map which map vertex in $G_{raw}$ to vertex in $G$
 4: for $v_{raw} \in vertices(G_{raw})$ do
 5:     $M := getExternalMethodCallSeq(v_{raw})$
 6:     for $m_i \in M$ do
 7:         $addVertex(G, m_i)$
 8:         if $i = 0$ then
 9:             $M_{head}[v_{raw}] = m_i$;
10:         else if $i = |M| - 1$ then
11:             $M_{tail}[v_{raw}] = m_i$;
12:         end if
13:         if $i > 0$ then
14:             $addEdge(G, m_{i-1} - > m_i)$;
15:         end if
16:     end for
17: end for
18: for $e_{raw} \in edges(G_{raw})$ do
19:     $N_{head} := findHeadNodes(inVertex(e_{raw}), M_{head}, )$;
20:     $N_{tail} := findTailNodes(outVertex(e_{raw}), M_{tail}, )$;
21:     for $n_{tail} \in N_{tail}$ do
22:         for $n_{head} \in N_{head}$ do
23:             $addEdge(G, n_{tail} - > n_{head})$;
24:         end for
25:     end for
26: end for
27: return $G$;
```

This compare algorithm AllCSofGraphComparePattern(All common Subsequence compare with pattern)7 will use in our analysis in two different stages, we will first apply this algorithm to compare our reconstructured subroutine CFGs graph with our checking pattern, and second we will apply this algorithm to compare our reconstructured

Algorithm 4 ReconstructureFCG(G_{fcg}, $sbrtAllCSs$)

```
Require: $G_{fcg}$: The function calls graph .graphml file of an iOS app $D_{sbrtAllCSs}$: A dictionary of LCS results list between pattern and all subroutines in app.
Ensure:
$G := \{\}$;                                        ▷ empty result graph
$M_{head} := \{\}$;                                 ▷ empty map which map vertex in $G_{raw}$ to vertex in $G$
$M_{tail} := \{\}$;                                 ▷ empty map which map vertex in $G_{raw}$ to vertex in $G$
for $E_{sbrt} \in D_{sbrtAllCSs}$ do
    $sbrtTitle := key(E_{sbrt})$;
    $L_{methodsSeq} := value(E_{sbrt})$;
    for $L_{methods} \in L_{methodsSeq}$ do
        $L_{path} = ;$
        for $method \in L_{methods}$ do
            $add(L_{path}, method)$
            $addVertex(G, method)$
        end for
        for $method_i \in L_{path}$ do
            if $i = 0$ then
                $M_{head}[v_{raw}] = m_i$;
            else if $i = |M| - 1$ then
                $M_{tail}[v_{raw}] = m_i$;
            end if
            if $i > 0$ then
                $addEdge(G, m_{i-1} - > m_i)$;
            end if
        end for
    end for
end for
for $e_{fcg} \in edges(G_{fcg})$ do
    $N_{head} := findHeadNodes(inVertex(e_{fcg}), M_{head}, )$;
    $N_{tail} := findTailNodes(outVertex(e_{fcg}), M_{tail}, )$;
    for $n_{tail} \in N_{tail}$ do
        for $n_{head} \in N_{head}$ do
            $addEdge(G, n_{tail} - > n_{head})$;
        end for
    end for
end for
return $G$;
```

FCG(function call graph) with our checking pattern.

First in algorithm8, we pair up all sink nodes(no incoming edge) and terminal nodes(no outgoing edge) in graph. Then, we take each pair as a path initial node and terminal node. Then, find out all the possible directed paths for each pair within current path length limit. Second we compute the AllCS(all common subsequence) between all the possible paths from the graph and our pattern sequence, and the list of all common subsequence will be the result we are checking for.

Algorithm 5 findHeadNodes(v_{raw}, M_{head}, $V_{visited}$)

```
Require: $v_{raw}$: The control flow graph(.graphml) of a subroutine within an iOS app. $M_{head}$:
Ensure:
if $v_{raw} \in V_{visited}$ then
    $return$;
end if
$add(V_{visited}, v_{raw})$;
if $v_{raw} \in M_{head}$ then
    $returnM_{head}[v_{raw}]$;
end if
$result := ;$
for $v_{in} \in inVertices(v_{raw})$ do
    $result := result \cup findHeadNodes(v_{in}, M_{head}, V_{visited})$;
end for
return $result$;
```

Algorithm 6 findTailNodes(v_{raw}, M_{tail}, $V_{visited}$)

```
Require: $v_{raw}$: The control flow graph(.graphml) of a subroutine within an iOS app. $M_{tail}$:
Ensure:
1: if $v_{raw} \in V_{visited}$ then
2:     $return\{\};$
3: end if
4: $add(V_{visited}, v_{raw});$
5: if $v_{raw} \in M_{tail}$ then
6:     $returnM_{tail}[v_{raw}];$
7: end if
8: $result := \{\};$
9: for $v_{in} \in inVertices(v_{raw})$ do
10:     $result := result \cup findHeadNodes(v_{in}, M_{tail}, V_{visited});$
11: end for
12: return $result;$
```

In algorithm7 we will try to find all the avaliable paths in graph and compare them with our checking pattern sequence. We will start with a initial number(number of graph vertexes) as our path length limit, and increase the number and reapply in the comparison until there has no more results show up.

Algorithm 7 AllCSofGraphComparePattern(G, $L_{pattern}$)

```
Require: $G$: A reconstructured subroutine CFGs graph or a reconstructured function call graph. $L_{pattern}$: A checking methods list compose
    a specific behavior or property.
Ensure:
1: $limit := numOfVertex(G);$
2: $preResultSize = 0;$                                  ▷ The result size for previous match result
3: $result := \{\};$
4: while true do
5:     $L_{path} := getAllDistinctPaths(limit, G);$
6:     for $P \in L_{path}$ do                           ▷ $P$ is collection of method sequence list
7:         $L_{allCS} := findAllCS(L_{pattern}, P);$
8:         $result := result \cup L_{allCS};$
9:     end for
10:     if $preResultSize = size(result)$ then
11:         $break;$
12:     else if
13:         then$preResultSize := size(result);$
14:         $limit := limit + 1;$
15:     end if
16: end while
17: return $result;$
```

3.4 Checking deprecated APIs

To preserve simplicity and usability in the long term, sometimes it is necessary to make changes to the API that will require changes to the applications using the guest application APIs. When this happens, the Apple Inc. will notify developers by publishing breaking changes in their release notes[3], and they will minimize disruptions by supporting the deprecated functionality for one major release.

Therefore, during the iOS SDK version updating, Apple Inc. will mark some methods as deprecated and these methods will be eventually suspended in the future. Although

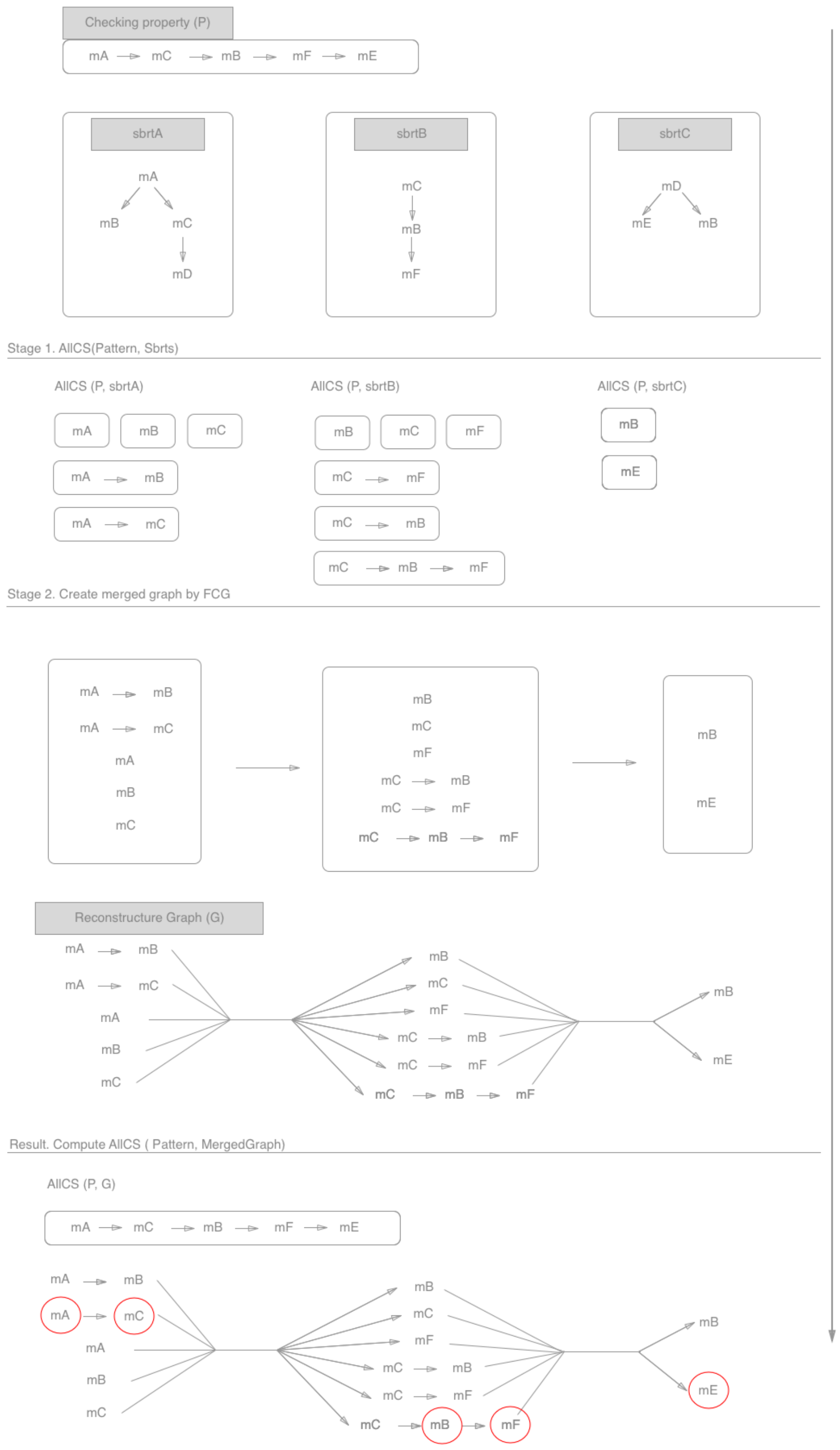


Figure 4: Two stages All common subsequence checking processing.

Algorithm 8 getAllDistinctPaths($limit$, G)

```
Require: G: A reconstructured subroutine CFGs graph or a reconstructured function call graph. limit: The limit path length of getting all
    distinct paths within graph
Ensure:
1:  $S_{sinkNodes} := \{\}$;
2:  $S_{terminalNodes} := \{\}$;
3:  for $v \in getAllVertices(G)$ do
4:      if $incomingEdgesOf(v) = 0$ then
5:          $add(v, S_{sinkNodes})$;
6:      end if
7:      if $outgoingEdgesOf(v) = 0$ then
8:          $add(v, S_{terminalNodes})$;
9:      end if
10:
11: end for
12: if $size(S_{sinkNodes}) = 0$ then
13:     $add(getAllVertices(G)[0], S_{sinkNodes})$;
14: end if
15: if $size(S_{sinkNodes}) = 0$ then
16:     $add(getAllVertices(G)[1], S_{terminalNodes})$;
17: end if
18: $result := \{\}$;
19: for $n_{sink} \in S_{sinkNodes}$ do
20:     for $n_{terminal} \in S_{terminalNodes}$ do
21:         $L_{allPath} = getAllPaths(n_{sink}, n_{terminal}, limit)$;
22:         $result := result \cup L_{allPath}$;
23:     end for
24: end for
25: return $result$;
```

the deprecated methods still supported by SDK now, Apple Inc. suggests developers should not use these deprecated methods for developing products and use the alternative solutions. When a method was marked as deprecated, it usually means that there was a replacement for a better solution, and the deprecated method should avoid to use.

Mostly the new methods will provide a better performance in the new operating system or some language features. We have collected all deprecated methods in our system and we will use the deprecated method list to check the deprecated API usages in the apps we download from AppStore. Besides, because the official document in iOS developer API reference[2] doesn't maintain a fully list for all the deprecated APIs, they only keep the change logs in each version release notes[3], we also maintain a list for all the deprecated APIs in iOS SDK.

When developers use iOS SDK to develop their programs, it is very important to follow the release notes and update their programs. Because the lack of updating may lead to programs will not support to client device, even crush the application executing. All of these issues, may cause the low user experience of the application, and even some security issue will happen.

To check these deprecated api usage in applications, we first generate the deprecated api method list in iOS SDK by our web crawler. Then, by using the application property existence checking, we can find out the usage of deprecated APIs within application.

3.5 Checking golden rules

In the iOS developer API reference[2], the Apple Inc. has provided some suggestions, example code or demonstrations for some scenario. For example, in iOS the biometric functions are mostly related to the Touch ID functionalities, when a developer wants to implement some biometric function in their applications, they have to implement some specific methods to get the authorization. We also collected these samples and suggestions, then translate them to a sequential method calls as a behavior property for checking. To check these golden rules, we have to use the sequential checking with LCS, and we will use these golden rules as our checking patterns. Then execute our sequential checking to compare these patterns with every application.

Recommend uses case

For the detail show as below:

1. Core Location: The framework let developers determine the current location or heading associated with a device, it also can define geographic regions and monitor when the user crosses the boundaries of those regions. When we start to access user's device location, we have to ask for user authorization through CLLocationManager class, the method calls show below:
   (a) CLLocationManager alloc
   (b) CLLocationManager init
   (c) CLLocationManager requestWhenInUseAuthorization
2. JavaScriptCore: The framework help developer evaluate JavaScript programs within apps, and support executing javascript in applications. When we need to evaluate a javascript programs, we have to perform these method calls in JSContext class:

   (a) JSContext init
   (b) JSContext evaluateScript

3. WebKit: The webkit framework implemented the browser features, and provide developer to display web content in windows, and the WKWebView object displays interactive web content, let developer can use the WKWebView class to embed web content in app. To do so, we have to create a WKWebView object, set it as the view and send it a request to load web content, the following show method calls:
   (a) WKWebView alloc
   (b) WKWebView init
   (c) WKWebView load

4. LocalAuthentication: The framework provide developer for requesting authentication from users through passphrases or biometrics. To do so, by using LAContext class.
   (a) LAContext alloc
   (b) LAContext init
   (c) LAContext canEvaluatePolicy
   (d) WKWebView evaluatePolicy

4 AppScan System

4.1 System Structure

To build up our experiment, a automatic system is necessary, therefore, we develop this system for application sequence call analysis (abbrv. on AppScan), as shown in figure 5. We make each step in our system are more flexible and extendable, and AppScan are designed like a production line, it can auto fetch new data, process our experiment tasks, collect the results and store then present. The whole system can be divided into three

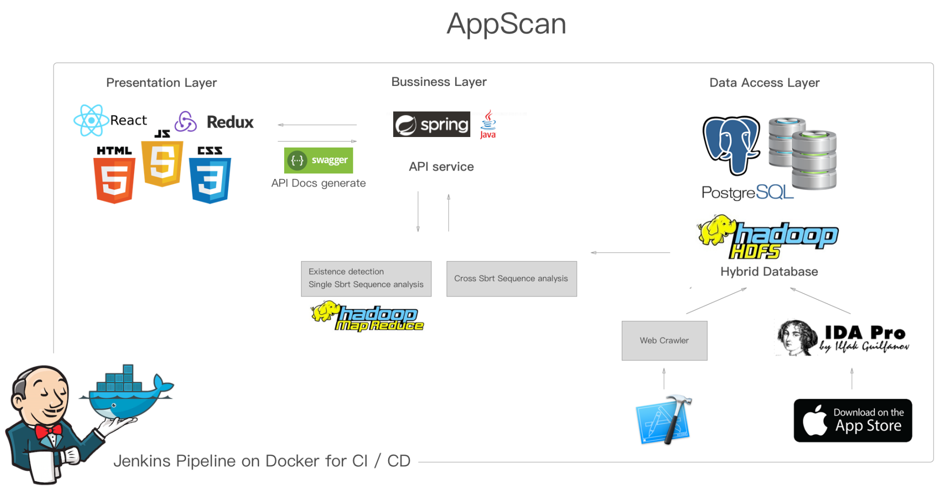


Figure 5: This structure show the whole system data flow and used tools and tech.

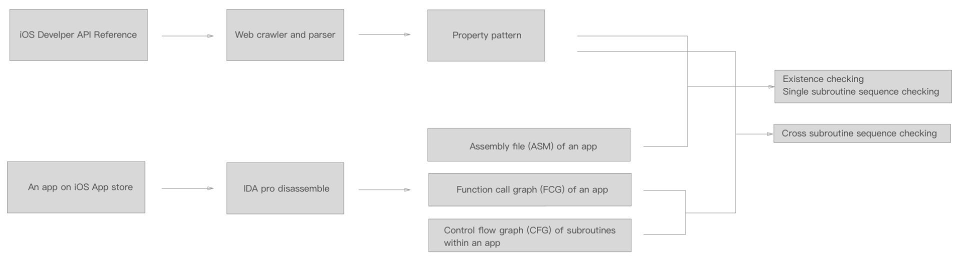

iOS Develper API Reference
Web crawler and parser
Property pattern
Existence checking
Single subroutine sequence checking
Cross subroutine sequence checking
Assembly file (ASM) of an app
An app on iOS App store
IDA pro disassemble
Function call graph (FCG) of an app
Control flow graph (CFG) of subroutines within an app

Figure 6: This flow show the whole system logic flow and processing sequence.

layers to interpret, data access layer, business logic layer and presentation. These arrows represent the data flow between different task in this system, and in the figure6 represent the system logic and processing flow. We first download apps form Apple App Store. The binary data are sent into IDA pro for converting into assembly files, and the metadata of an app will be store in our database. Then, in one side we will use a web crawler to fetch our method list in iOS Developer API Reference site. The list and the input property pattern subsequence will be conduct the syntax check with hadoop map-reduce structure. By running syntax sequence matching with the list, we can infer that an app has the property or not. All these results will be stored in our database. Then we will construct a API service for accessing these results, and preform these results in web application form with front-end library react, flux, and visualization library d3js.

4.2 Research and system process

4.2.1 Data collection and processing

Sandbox of iOS and binaries on App Store

When an iOS app finished, the developers have to compile and submit it to App Store for publishing. We can easily download these binaries with our iDevice or iTunes application in our computer then synchronize the binaries with our iDevice. However, Apple. Inc. set a sandbox policy of iOS. It means that, one application is restricted in an container, and granted limited resources under its own directory, and many important features of an operation system are forbidden in iOS. For building an stable and robust system environment to users is a good policy, but it also restricted users' right to access this system file in common way. In order to make our experiment workable, we have to get the highest permission of iOS system, generally speaking we call this jailbreak.

Auto download apps from App Store

At beginning of this system, the apps' binaries must be well prepared. So, we develop a auto download process by using Selenium[28], a browser driver, and Sikuli[9], a GUI control library using image recognition to identify any component on screen. With Selenium, we can drive the web browse to the application download link page on Apple app store, and automation click download button to redirect and active the iTunes on our iMac. Then, jump out the browser environment, we will use Sikuli to control our iTunes application on macOS and download the app and synchronize into our connected iPhone.

Decrypt and extract application from iPhone

When we have binaries in iPhone, we have to dump them into our computer for decrypt and analysis. Because the binaries of apps represent in digital(0 or 1), that is hard to comprehend for people. We have to resolve these binaries into assembly files further. Here is a quick overview for whole process of application binaries, and I will go through the details below. Because all available apps on App Store are singed and encrypted by

Apple Inc., until these binaries execute on iDevice. When we got these binaries, we have to decrypt first then we could analysis them. In this step, we have jail broken our iPhone to have the highest permission. The current active jail break tool can help us to get this permission under the version 9.0.2 of iOS operation system. After we got an jail broken device, we got sufficient right to access the device' file system and use sftp (SSH File Transfer Protocol) with scp command to dump these apps to our computer. To achieve the goal, we have find the entry point within this binary then execute it under debugger and dump the decrypted part and patch the origin encrypted binary. After doing these steps, we will get an decrypted binary. Next, we use disassembler tool IDA pro[29], to help us convert these binaries to assembly files. Once we got these assembly files, we can start our analysis on the context later.

Decrypting Apps

Installed mobile application on iOS device is accompanied with its executable, databases, media data files, and they would be placed within a specific file directory (so-called sandbox). At normal situation, users can't get permission to access these file directories. To extract the binary file of applications for our research, we break the sandboxes and act as root administrators with the highest privileges. Then, we could access application file directory to retrieve data. Nowadays, there are lots of jailbreak tool for different iOS versions on the Internet. We use the jailbreaking tool for an iOS device is PanGu9, which can jailbreak iOS version from 9.0 to 9.0.2 [4]. After finishing the jailbreak process, we could find an icon of Cydia, a third party iOS application repository, on the iDevices desktop. We download OpenSSH and MobileTerminal packages, which enable us to connect to the iDevice as root administrator by using ssh tunnels from Cydia. All of the installed applications from AppStore in iDevice are stored under a director, /var/mobile/Containers/Bundle/Application, where we use ssh command to access them after the jailbroken. The apps downloaded from Apple App store are partially encrypted to prevent official apps from third-party disassembling and reverse engineering. To overcome these obstacles, we leverage the property of debugging technique, i.e. and dumping decrypted machine instructions during the runtime, since it's always decrypted before execution, and swapping the encrypted part with the decrypted one and the decrypted version is obtained. We use third-party library offered by Stefan Esser and adjust library to apply to iOS 9 [5]. With the library, we could dump out decrypted files from encrypted applications. We write a python script with his library so Mach-O application files on an iDevice could be decrypted and transfer files to an external storage.

Disassembling App

After retrieving application files, we disassemble applications to generate assembly language source code and related information from machine-executable code in this section. We could do static binary analyzing on assembly code in following sections. In this procedure, we rely on Interactive Disassembler, aka IDA, a powerful multi-processor disassembler, and debugger to extract related information from compiled executable. IDA has two different versions, the starter and the pro. The former only supports process 32-bits executable, while the latter is capable of processing 32bits and 64-bits executable. We also install IDAPython, an extension plugin for IDA, allowing python script to run in IDA environment. We use IDA pro version and write a python script to unveil all the hidden process, and then generated an ASM file, which is written in assembly language, control flow graph of all subroutines, and function call dependency graph.

4.2.2 Application methods property list and iOS API reference web crawler

In our system, we want to check whether given property appear in an application or not, so we have to prepare a list of all possible methods in iOS SDK to compose that property. Because most of developers develop iOS application with objective-C language, and there are also many SDKs (Software Development Kit) for objective-C provided by no matter officially Apple Inc. To get all these methods data, we directly develop a web crawler for the iOS API Reference site hosted by Apple Inc. After fetching all the information we need, we can generate the whole framework class and method list. Therefore, we can index these methods, classed and frameworks for easier processing and storing in our database.

At the same time, we can also generate property pattern from these methods we fetched for specific application behavior. For example, we may interest in an application asking for the authority of private data access function. Then we can use the these methods to generate the property. As we mentioned early, we use these methods to generate the deprecated API list, and the golden rule properties.

4.2.3 Property syntax sequence processing

Now, we have well-prepared all the data we need to process. We can use hadoop to execute the analysis for existence checking and single subroutine sequential checking, and use CFGs, FCG with two stage AllCS method for acrossing subroutines sequential checking. First, we will compile many jar files for different purpose and analysis as we mentioned before. Substantially, for one purpose of analysis will have a corresponding jar. For example, we have a jar for checking deprecated API, and a jar for check LCS sequence with Location property ..., etc. Therefore, we can chain all our analysis processes together, and log each step in our database. Basically, for every task in our system, it will need a target application, a analysis property pattern, and a correspond jar file as inputs. By different analysis task and purpose, we will maintain different job queue to monitor all the process. Here, we will initial a executor to dispatch new task in to our jobs queue, and monitor our pending jobs and completed jobs in database, and change the analysis status for each job target.

4.2.4 Web service

Back-end API Service - Restful architecture

To automate the system process when we have any new data input, we make our system can record the whole experiment work flow and record the status. Building it up with Java Spring framework with postgresSQL and HDFS. However, directly manipulate these file in HDFS or data in database is very tedious and error prone. So we mapping the database schema into program objects, by manipulate these objects, we can relatively easy to use these data. Next, we wrapped our results into data endpoints and export APIs in RESTful style to make them more clearly, comprehensive, and easy to communicate with front-end application.

ORM, ODM and HDFS

Our database are designed for many sequential jobs for different input data, and we adopt different features in different database types, and further make a hybrid database system with relational database, and HDFS(Hadoop File System). We use RDB(Relational Database) with ORM mapping tool to record our jobs status and work-flow, and store results as JSON blob type. The others raw data such .asm files or image files ... , we will keep these blob data in HDFS and record the URI in our RDB, that will help us easy to find and access these data.

Front-end presentation and data visualization

Because most of our experiment results are not easy to realize in short time. Making our result more easy to read, we use many charts and tables with visualization tool to present our result. On the other side, to solve the presentation of complicate data form different experiment results, we use ReactJS library with data flow architecture redux to make whole web application more maintainable.

ReactJS, Redux

ReactJS is an open source project for building web application provided by Facebook.Inc . In our implementation, we use this library to present our whole system. ReactJS provide many convenient feature to make front-end application more flexible, for example, we make each type of our experiment result into a basic present unit and wrap it into an component in React, by reusing and assembling these components we can quickly finish our web page. In the same time, we adopt Redux, an application architecture, to make our front-end web application more organized and maintainable. Redux provide an single way data flow with a state control center idea, that is, we will maintain all data in a data store with many different defined states, and by dispatching different actions to change data state. When data state changed, it will trigger the render component to update the view if necessary. By this way, we will make our data in a single way data flow cycle that can make data more easy to handle its state and trace logic bugs. Besides, to make our experiment results more comprehensive, we use many visualization tool for data presentation and make them more readable and meaningful.

5 Experiment Analysis

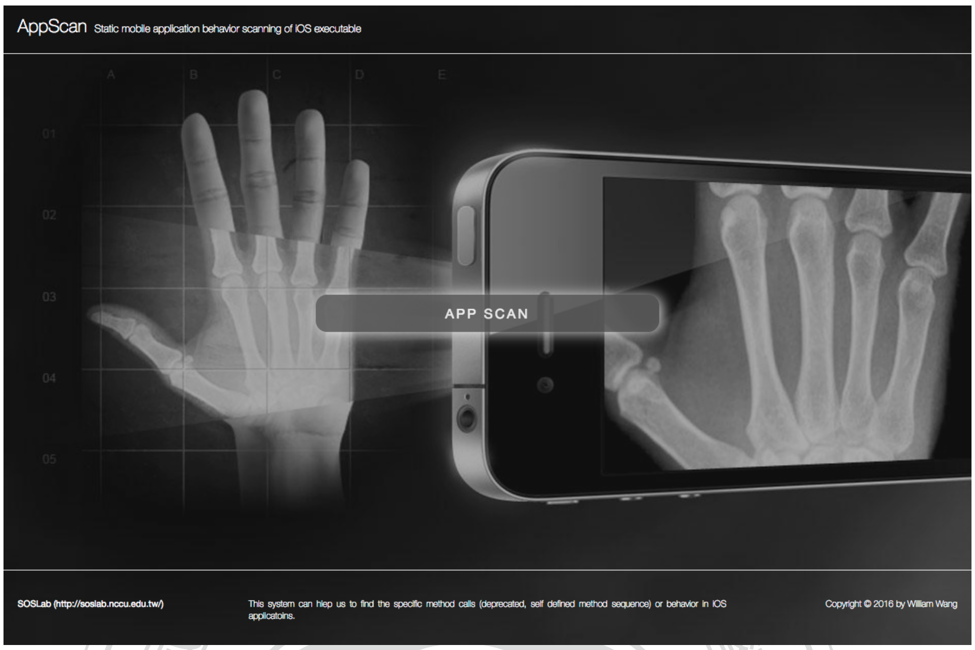
5.1 Intro

In this research, we have downloaded around 13000 apps and successfully disassembled around 2200 apps shown in Figure8 available on AppStore, and still downloading currently. The following statistic results and founds are based on these apps that we successfully processed. We major analysis is on the deprecated usage analysis and golden rules pattern checking, and the data will be present and downloadable on the website AppScan shown in Figure 7 we hosted.

5.2 Usage of Deprecated APIs Result

We use our system to analysis the apps that were successfully disassembled in total, and we successfully analysis 1505 apps shown in Figure 9. In our results, we found that there are 1366 apps have used the deprecated APIs and available for downloading on AppStore.

Further, our analysis results show most of usage of deprecated APIs are distributed in the UIKit, and the Foundation framework, as shown in the Figure 10. The UIKit and Foundation frameworks are very important frameworks in iOS SDK. The UIKit are the abbreviation of User Interface Kit, so the methods under this framework provided are

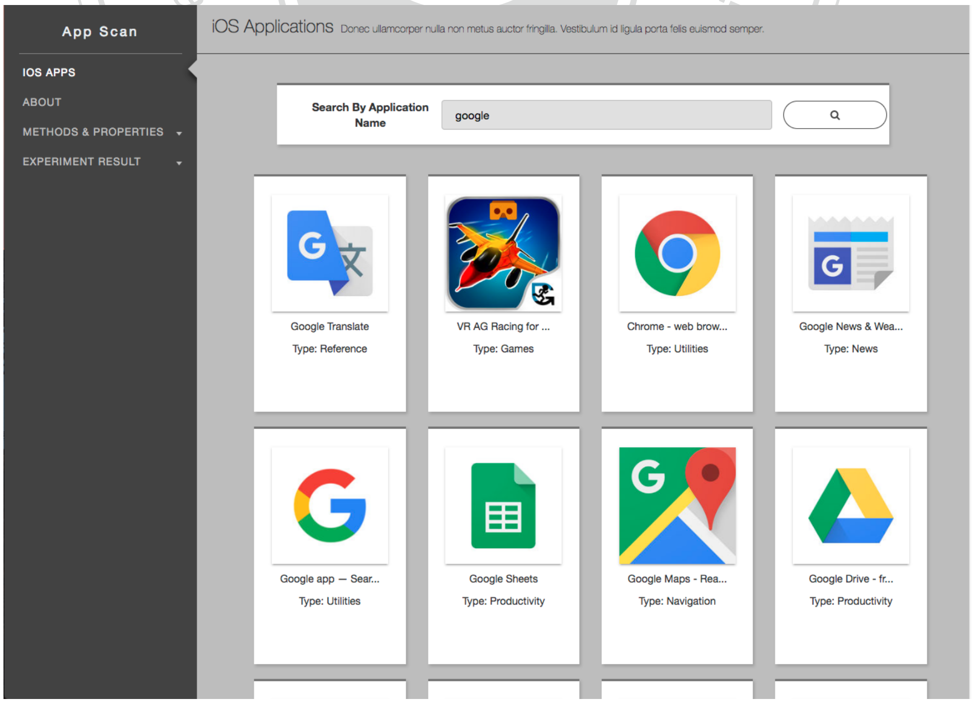

AppScan Static mobile application behavior scanning of iOS executable
A
B
C
D
E
01
02
03
04
05
APP SCAN

SOSLab (http://soslab.nccu.edu.tw/)

This system can hiep us to find the specific method calls (deprecated, self defined method sequence) or behavior in iOS applicatoins.

Copyright © 2016 by William Wang

Figure 7: Landing page of website AppScan.



iOS Applications Donec ullamcorper nulla non metus auctor fringilla. Vestibulum id ligula porta felis euismod semper.

Search By Application Name: google

Google Translate
Type: Reference

VR AG Racing for ...
Type: Games

Chrome - web brow...
Type: Utilities

Google News & Wea...
Type: News

Google app — Sear...
Type: Utilities

Google Sheets
Type: Productivity

Google Maps - Rea...
Type: Navigation

Google Drive - fr...
Type: Productivity

Figure 8: Apps we downloaded and disassembled from AppStroe.

related to user interface. It provided to construct and manage an app's user interface for iOS. Respond to user interactions and system events, access various device features, enable accessibility, and work with animations, text, and images. The Foundation is the base layer of Objective-C classes. In official documentations, says that this framework is designed with several goals :

1. Provide a small set of basic utility classes.
2. Make software development easier by introducing consistent conventions for things such as deallocation.
3. Support Unicode strings, object persistence, and object distribution.
4. Provide a level of OS independence, to enhance portability.

Because the methods provided by these two frameworks are very generally and fundamentally, we need to go further to find what are the purpose of deprecated methods are using in applications. Then we dig into the classes and that used in theses frameworks as shown in Figure11 and Figure13, besides, we also sort the all the deprecated methods calls in applications, and list the top ten in Figure12. From these distribution charts, we can easily found that the most of uses methods in UIKit are related to the classes of UIAlertView and UIApplication, and we pick some of applications from it, and show the more detail usage in Figure 1415, and 16 for comparing.

The UIAlertView is using to display an alert message to the users, and all this class and the related protocol (such as the UIAlertViewDelegate) were deprecated in iOS 8. To create and manage alerts in iOS 8 and later, instead use UIAlertController with a preferredStyle of alert. The UIApplicatoin class is not a deprecated class like UIAlertView. It is design to provides a centralized point of control and coordination for apps running in iOS. Almost every app has one instance of UIApplicatoin. When an app is launched, the system will call the UIApplicationMain function and create a singleton UIApplication object and provide mothods for developer can access the object by the class method

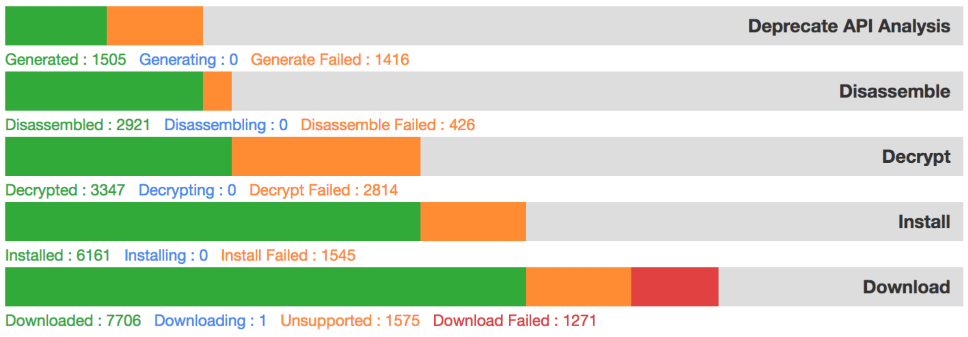


Figure 9: The job progressing of deprecated API usage checking.

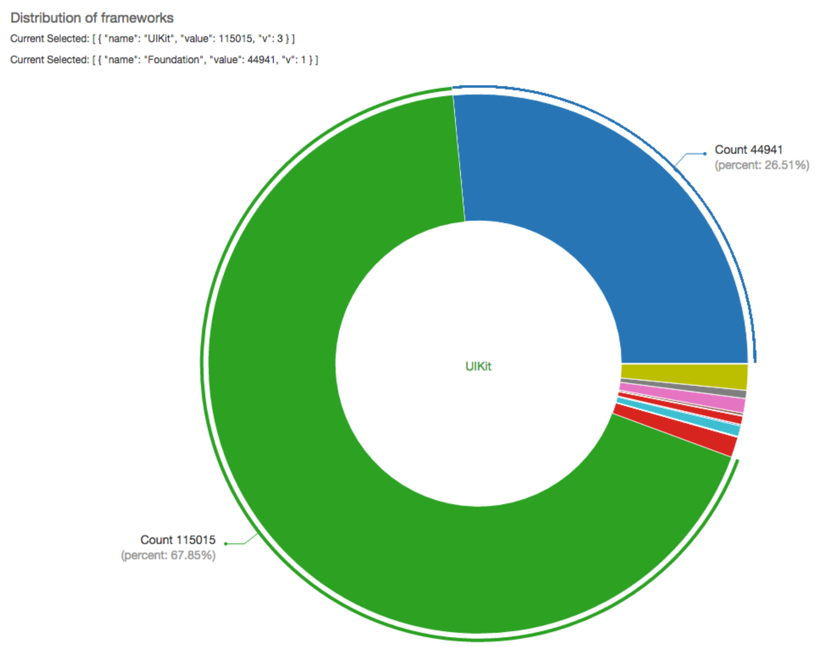
”sharedApplication”. However, it also has several outdate methods were marked as deprecated. From the Figure12, the ”openURL:” and ”setStatusBarHidden:withAnimation:” are the most uses in this class.

The ”openURL:” is use on open the resource at the specified URL, after deprecated, using the ”openURL:options:completionHandler:” method instead, and the ”setStatusBarHidden:withAnimation:” can hides or show the status bar, optionally animating the transition, after deprecated in iOS 9, and it can use a property to control the status bar ”prefersStatusBarHidden”, YES if the status bar should be hidden or NO if it should be shown. Although, these user interface related problems can be customized and bypass the problem, for example, we may create a custom view to replace the alert view provided by iOS SDK. However, from this result, most of applications still have the UIAlertView in their application, when it wants to display an alert message to the users, and the situation will face the problem when Apple stop support and remove the method form the iOS SDK in the future.

5.3 Syntax pattern check Result

We have four patterns defined in golden rules including ”evalJS”, ”getUserLocation”, ”getUserLAPermission” and ”loadByWKWebKit”, and we check 1322 applicatinos that were disassembled successfully, as shown in Figure 17.

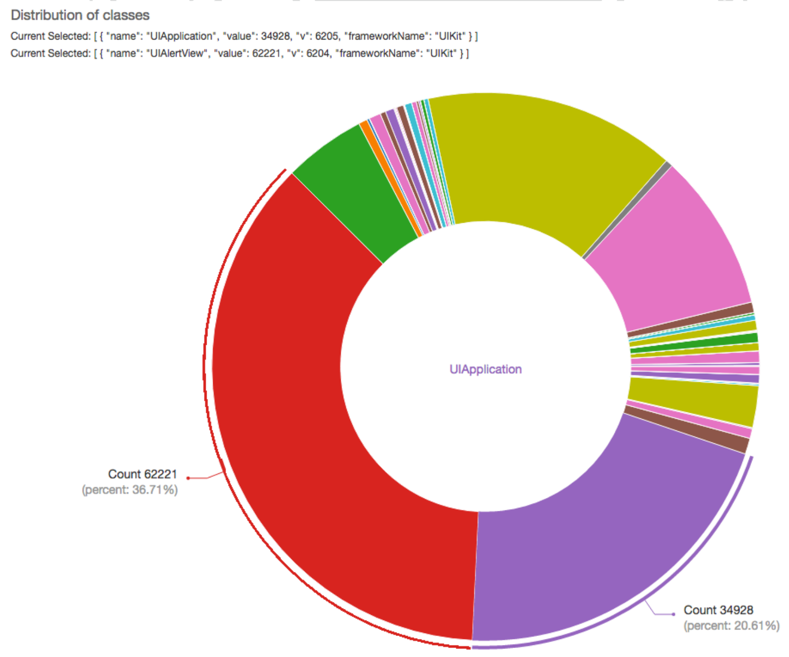
In the results of single subroutine checking, we can find out the applications that adopt the golden rules by checking the matching result counts, and we also collect the top 4


Distribution of frameworks

Current Selected: [{ "name": "UIKit", "value": 115015, "v": 3 }]

Current Selected: [{ "name": "Foundation", "value": 44941, "v": 1 }]

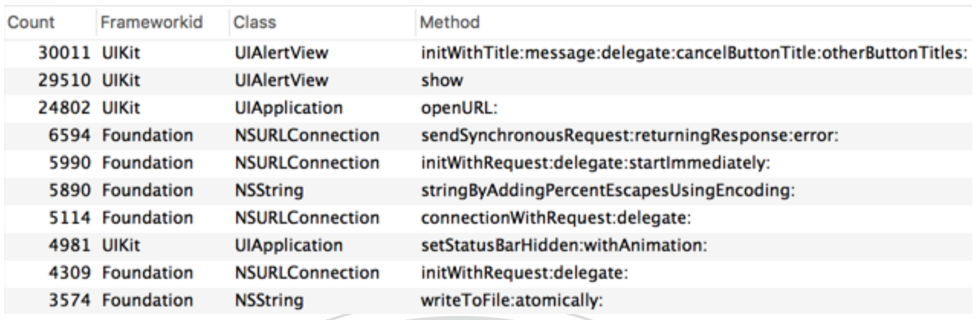
Figure 10: Deprecated APIs distrubution of iOS SDK frameworks.


Distribution of classes

Current Selected: [{ "name": "UIApplication", "value": 34928, "v": 6205, "frameworkName": "UIKit" }]

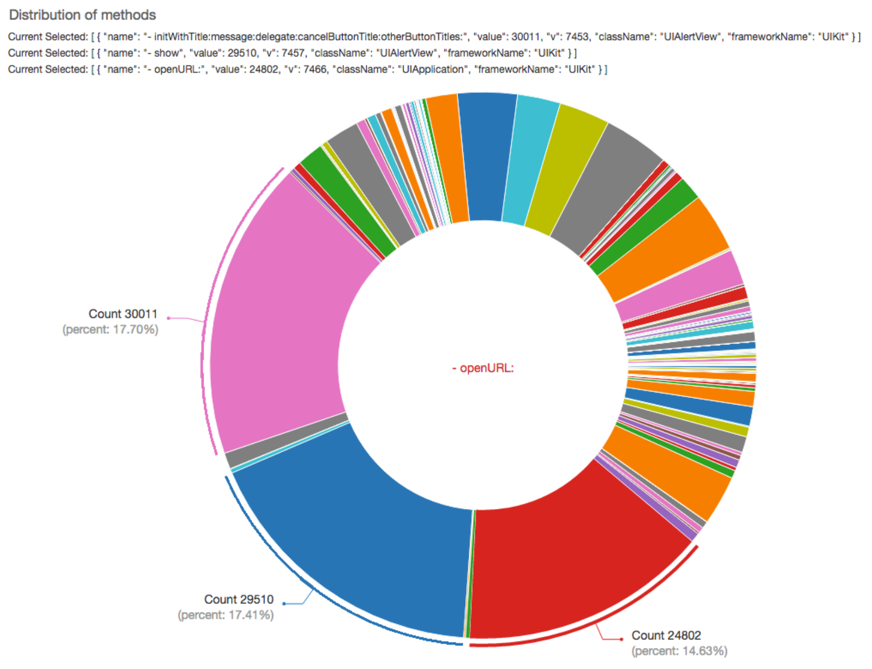
Current Selected: [{ "name": "UIAlertView", "value": 62221, "v": 6204, "frameworkName": "UIKit" }]

Figure 11: Deprecated APIs distribution of iOS SDK classes.



Count	Frameworkid	Class	Method
30011	UIKit	UIAlertView	initWithTitle:message:delegate:cancelButtonTitle:otherButtonTitles:
29510	UIKit	UIAlertView	show
24802	UIKit	UIApplication	openURL:
6594	Foundation	NSURLConnection	sendSynchronousRequest:returningResponse:error:
5990	Foundation	NSURLConnection	initWithRequest:delegate:startImmediately:
5890	Foundation	NSString	stringByAddingPercentEscapesUsingEncoding:
5114	Foundation	NSURLConnection	connectionWithRequest:delegate:
4981	UIKit	UIApplication	setStatusBarHidden:withAnimation:
4309	Foundation	NSURLConnection	initWithRequest:delegate:
3574	Foundation	NSString	writeToFile:atomically:

Figure 12: Top 10 usage deprecated APIs of iOS SDK methods.



Distribution of methods

Current Selected: [{ "name": "- initWithTitle:message:delegate:cancelButtonTitle:otherButtonTitles:", "value": 30011, "v": 7453, "className": "UIAlertView", "frameworkName": "UIKit" }]
Current Selected: [{ "name": "- show", "value": 29510, "v": 7457, "className": "UIAlertView", "frameworkName": "UIKit" }]
Current Selected: [{ "name": "- openURL:", "value": 24802, "v": 7466, "className": "UIApplication", "frameworkName": "UIKit" }]

Figure 13: Deprecated APIs distrubution of iOS SDK methods.

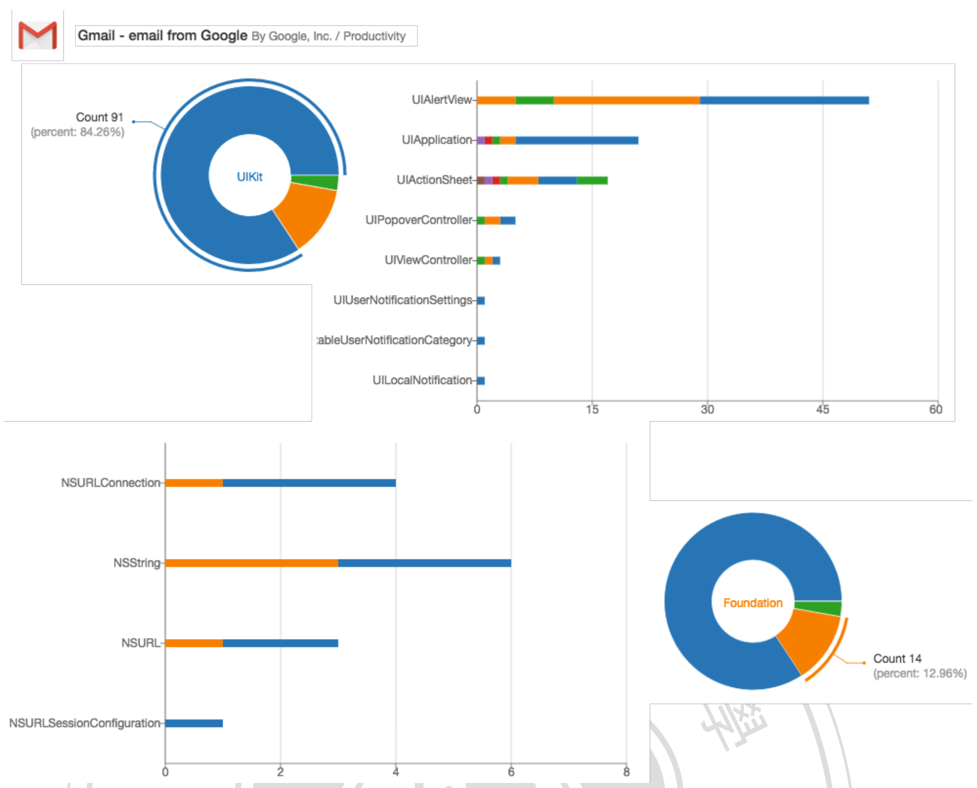


Figure 14: The distribution of deprecated APIs of Gmail.

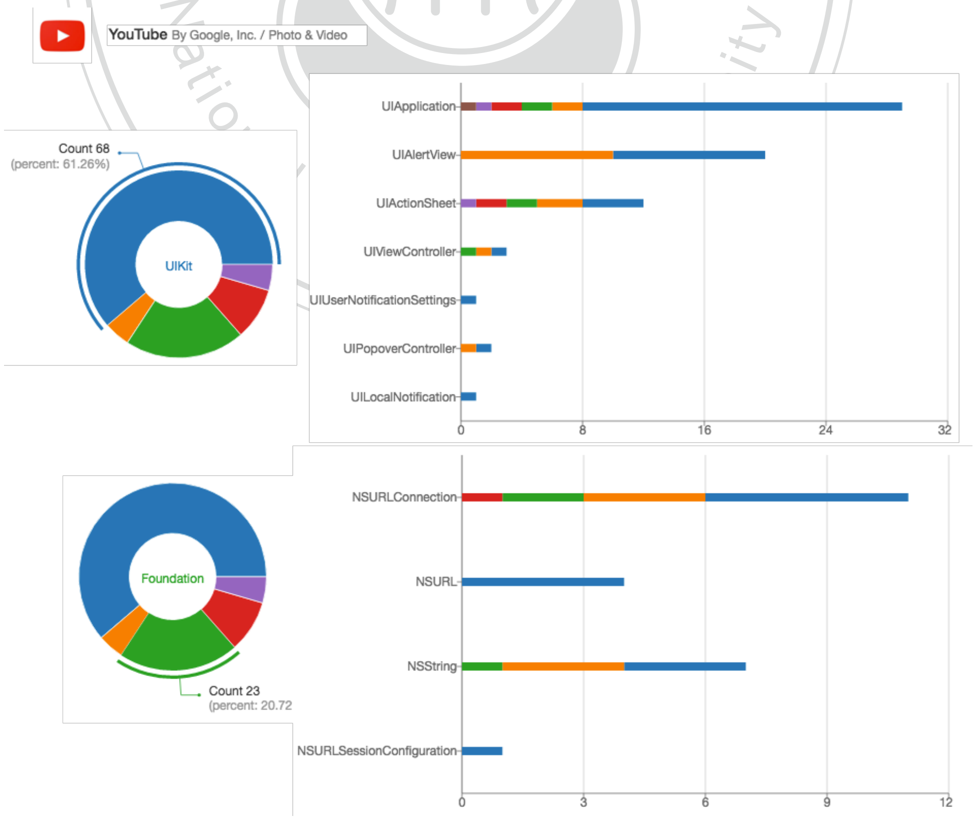


Figure 15: The distribution of deprecated APIs of Youtube.

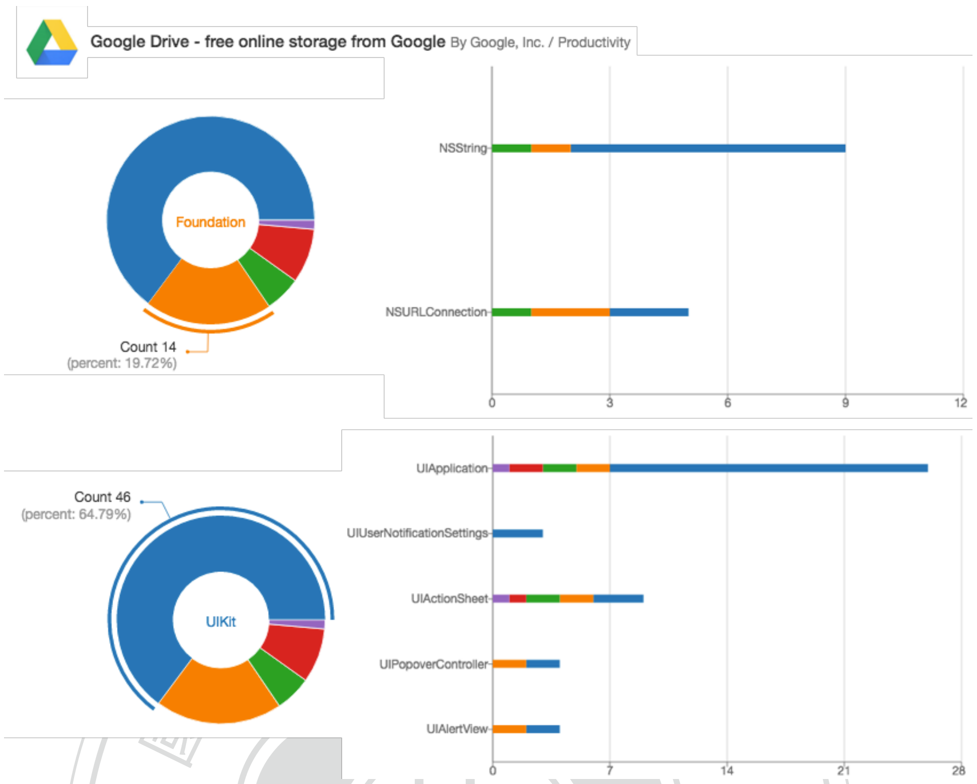


Figure 16: The distribution of deprecated APIs of GoogleDrive.

similar application in our results. For example, the result of top 4 in evalJS pattern check shown in Figure 18, and for the detail, we will show the checking result by LCS count, and the pattern methods matched in the application like Figure 19. Therefore, we can judge an application is following the pattern to executing or not.

However, the sequential checking with each single subroutine and may cause many missing. By using acrossing subroutines checking with two stage two stage all common sequence (AllCS). We have a rapid progress and judge application behavior more accurately. The results of compare two checking methods as show in the following figures 21 2224 ?? 20

In the research, we create an automatic system, AppScan. By tracing these graph, we can trace whole application function calls across different subroutines, and adopt the two stage all common sequence (AllCS) as our algorithm for checking application behavior. AppScan can analysis method sequence pattern in within an application. Therefore, we can define any pattern we want to check with applications. It greatly improves the

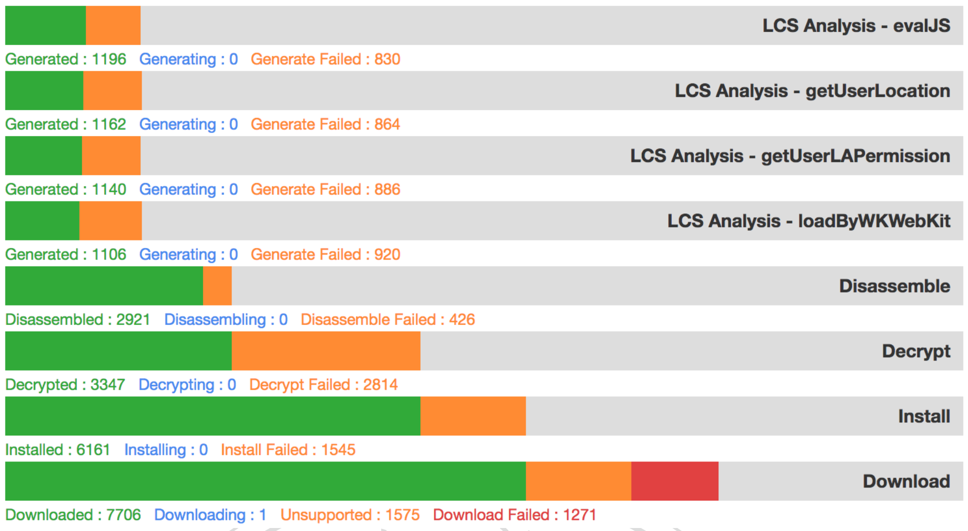


Figure 17: The job progressing of pattern checking.

sequence analysis method we used before. Curreltly we have checked the defined pattern "getUserLocation", and we have checked 1322 applicatinos that were checked successfully, as show in ??. The second and third column is application name and pattern name. The checking result is represented in a JSON array form with method names in the last column. This result can reveal that is an application use the right method to access user location authorization, and is the method is used in a proper way as Apple's suggestion.

For currently result, we can find out an application that adopt some feature functions but not follow the pattern or official guideline. When these situations occur, it means that the application should be improve or checking the risk of the unexpected method call sequence.

5.4 Conclusion and evaluation

In the research, we create an automatic system, AppScan. Developing three different checking methods for checking methods existence, single subroutine sequential checking with LCS, and across subroutines sequential checking with two stage AllCS. By using these methods we can checking application behavior by using our defined checking patterns.

When checking methods existence within an app, we using each single subroutine as our input unit. By filtering and compare the class name with the checking pattern, we

2. JavaScriptCore: The framework help developer evaluate JavaScript programs whithin apps, and support executing javascript in applications. When we need to evaluate a javascript programs, we have to perform these method calls in JSContext class:
(a) JSContext init
(b) JSContext evaluateScript

```
[
    {
        "className": "JSContext",
        "methodName": "init"
    },
    {
        "className": "JSContext",
        "methodName": "evaluateScript:"
    }
]
```

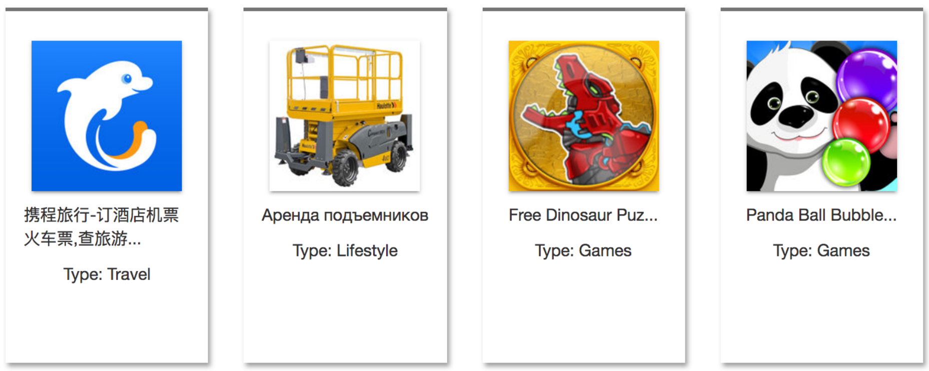
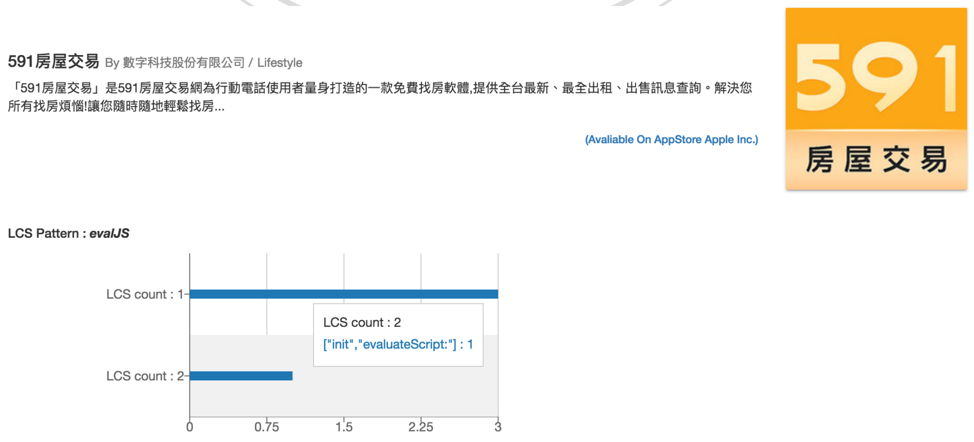


Figure 18: The top 4 of evalJS checking result.


591房屋交易 By 數字科技股份有限公司 / Lifestyle
「591房屋交易」是591房屋交易網為行動電話使用者量身打造的一款免費找房軟體,提供全台最新、最全出租、出售訊息查詢。解決您所有找房煩惱!讓您隨時隨地輕鬆找房...
(Avaliable On AppStore Apple Inc.)

LCS Pattern : *evalJS*

LCS count : 2
["init","evaluateScript:"] : 1

Figure 19: The evalJS LCS count result of an application.

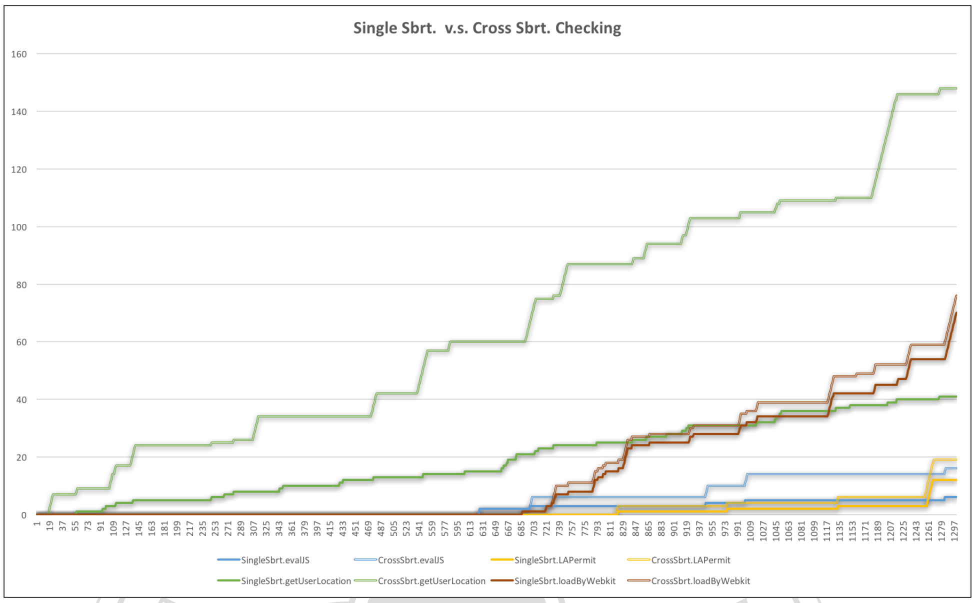


Figure 20: All patterns compare results between single subroutine and across subroutines checking methods.

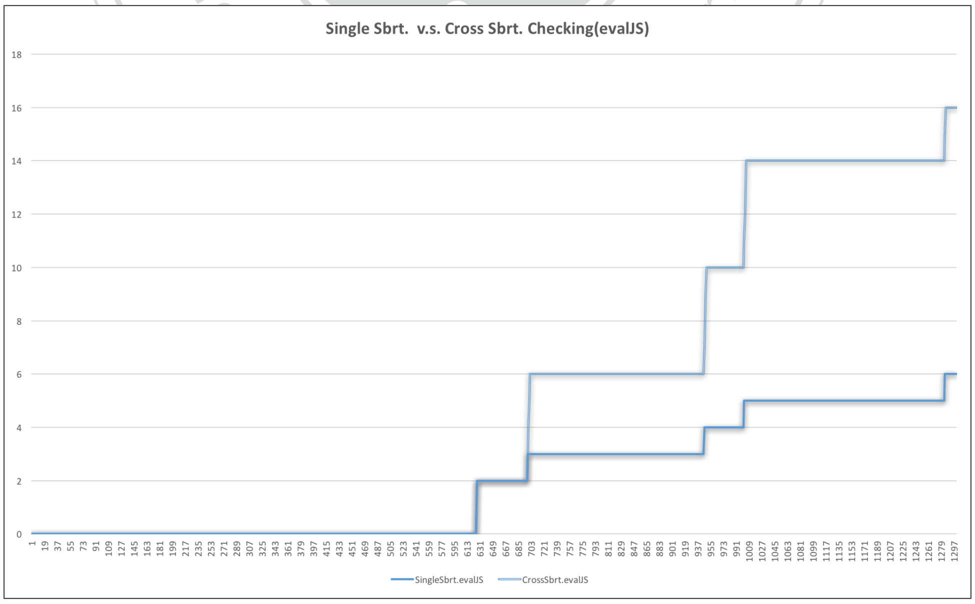


Figure 21: Javascript core pattern compare results between single subroutine and across subroutines checking methods.

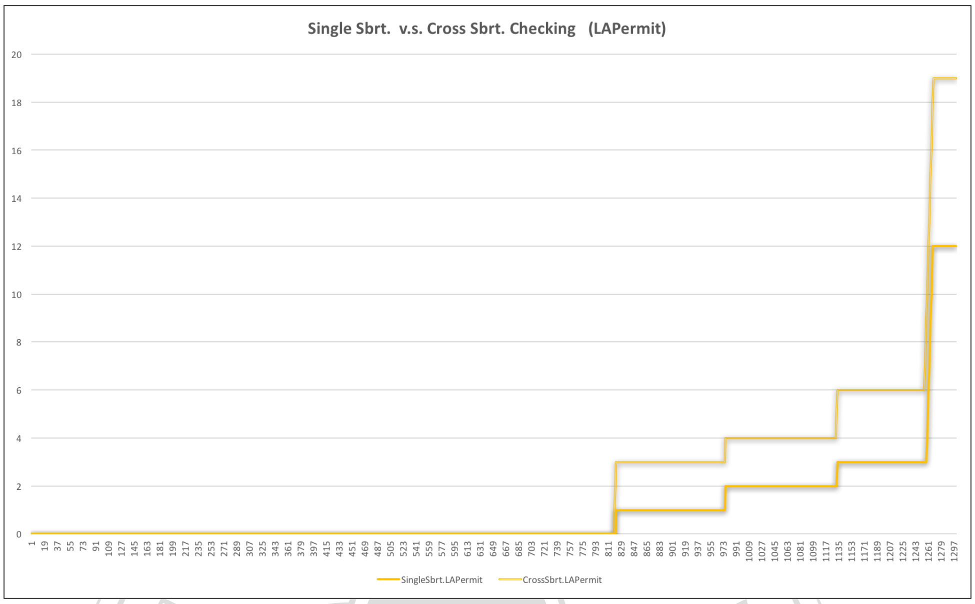


Figure 22: LAPermit pattern compare results between single subroutine and across subroutines checking methods.

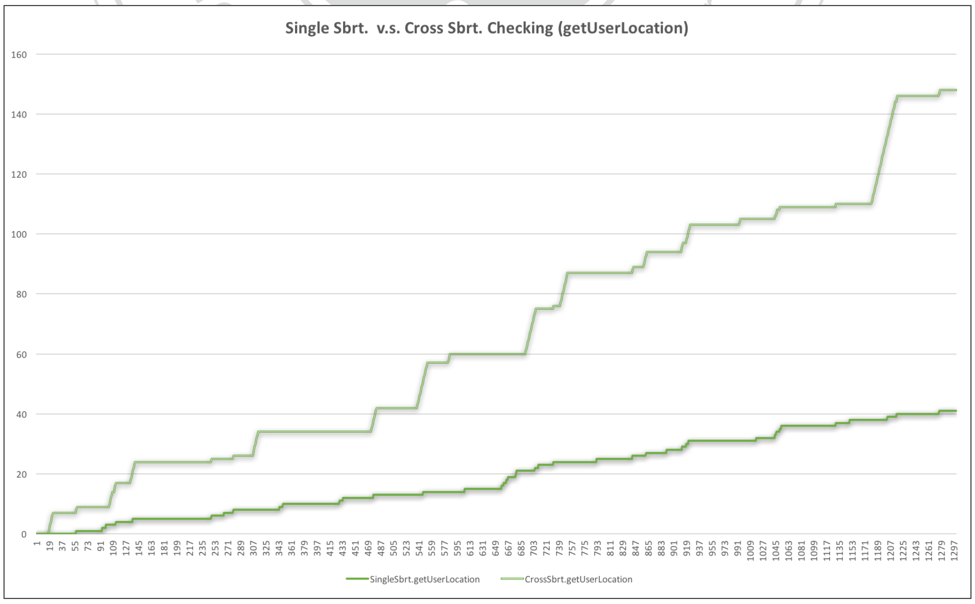


Figure 23: Get user location pattern compare results between single subroutine and across subroutines checking methods.

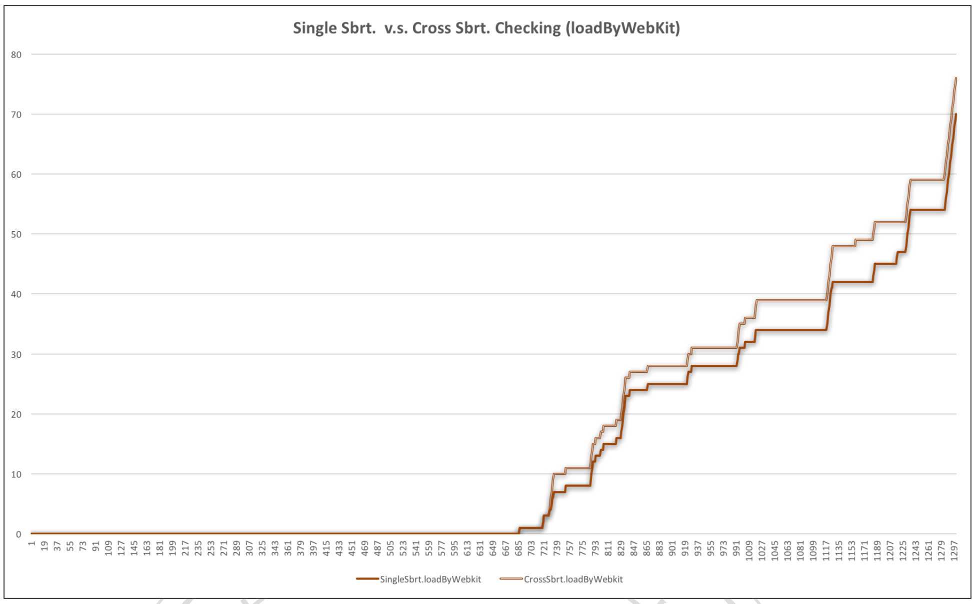


Figure 24: Load by Webkit pattern compare results between single subroutine and across subroutines checking methods.

can finially figure out all the checking method usage distribution infomation.

In sequential checking scenario, we adopt the distributed longest common sequence (LCS) as our algorithm at first. Using each single subroutine as our input unit. Because of the reasons, AppScan can analysis multiple method sequences pattern in each subroutine within an application. Therefore, we can define any pattern we want to check with applications. Next, we use two stage AllCS methods to improve the results, and we can check the pattern across multiple subroutines. It greatly improves the sequence analysis method we used before.

However, these solutions still has some limitations, for example, the system have to update at all times with Apple iOS SDK, and I hope that these limitations will be figure out in next progress.

References

[1] Apache hadoop. http://hadoop.apache.org/.

[2] ios developer api reference. https://developer.apple.com/reference/.

[3] ios release notes. https://developer.apple.com/library/content/releasenotes/General/WhatsNewIniOS/.

[4] Pangu ios 9. Available online at urlhttp://www.pangu.io.

[5] stefanesser umpdecrypted. Available online at url-https://github.com/stefanesser/dumpdecrypted.

[6] ios developer program license agreement. https://developer.apple.com/programs/terms/ios/standard/ios_program_standard_agreement_20140909.pdf, jan 2016.

[7] Yuvraj Agarwal and Malcolm Hall. Protectmyprivacy: detecting and mitigating privacy leaks on ios devices using crowdsourcing. In *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*, pages 97–110. ACM, 2013.

[8] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. *Acm Sigplan Notices*, 49(6):259–269, 2014.

[9] User Interface Design Group at MIT. Sikuli. http://www.sikuli.org/.

[10] Gleison Brito, Andre Hora, Marco Tulio Valente, and Romain Robbes. Do developers deprecate apis with replacement messages? a large-scale analysis on java systems. In *Software Analysis, Evolution, and Reengineering (SANER), 2016 IEEE 23rd International Conference on*, volume 1, pages 360–369. IEEE, 2016.

[11] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008.

[12] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[13] Zhui Deng, Brendan Saltaformaggio, Xiangyu Zhang, and Dongyan Xu. iris: Vetting private API abuse in ios applications. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-6, 2015*, pages 44–56, 2015.

[14] Adam Shook Donald Miner. *MapReduce Design Patterns*. O'Reilly Media, May 2012.

[15] Manuel Egele, Christopher Kruegel, Engin Kirda, and Giovanni Vigna. Pios: Detecting privacy leaks in ios applications. In *NDSS*, 2011.

[16] William Enck, Peter Gilbert, Seungyeop Han, Vasant Tendulkar, Byung-Gon Chun, Landon P Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N Sheth. Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. *ACM Transactions on Computer Systems (TOCS)*, 32(2):5, 2014.

[17] Adrienne Porter Felt, Matthew Finifter, Erika Chin, Steve Hanna, and David Wagner. A survey of mobile malware in the wild. In *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, pages 3–14. ACM, 2011.

[18] Yu Feng, Saswat Anand, Isil Dillig, and Alex Aiken. Apposcopy: Semantics-based detection of android malware through static analysis. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 576–587. ACM, 2014.

[19] Alessandra Gorla, Ilaria Tavecchia, Florian Gross, and Andreas Zeller. Checking app behavior against app descriptions. In *Proceedings of the 36th International Conference on Software Engineering*, pages 1025–1035. ACM, 2014.

[20] Jin Han, Qiang Yan, Debin Gao, Jianying Zhou, and Huijie Robert DENG. Android or ios for better privacy protection? 2014.

[21] Jianjun Huang, Xiangyu Zhang, Lin Tan, Peng Wang, and Bin Liang. Asdroid: Detecting stealthy behaviors in android applications by user interface and program behavior contradiction. In *Proceedings of the 36th International Conference on Software Engineering*, pages 1036–1046. ACM, 2014.

[22] Apple Inc. Apple worldwide developers conference 2015. `https://developer.apple.com/videos/wwdc2015/`, 2015.

[23] Mariantonietta La Polla, Fabio Martinelli, and Daniele Sgandurra. A survey on security for mobile devices. *IEEE communications surveys & tutorials*, 15(1):446–471, 2013.

[24] Li Li, Alexandre Bartel, Tegawendé F Bissyandé, Jacques Klein, Yves Le Traon, Steven Arzt, Siegfried Rasthofer, Eric Bodden, Damien Octeau, and Patrick McDaniel. Iccta: Detecting inter-component privacy leaks in android apps. In *Proceedings of the 37th International Conference on Software Engineering-Volume 1*, pages 280–291. IEEE Press, 2015.

[25] Li Li, Tegawendé F Bissyandé, Damien Octeau, and Jacques Klein. Droidra: Taming reflection to support whole-program analysis of android apps. In *Proceedings of the 25th International Symposium on Software Testing and Analysis*, pages 318–329. ACM, 2016.

[26] Benjamin Livshits and Jaeyeon Jung. Automatic mediation of privacy-sensitive resource access in smartphone applications. In *Proceedings of the 22th USENIX Security Symposium, Washington, DC, USA, August 14-16, 2013*, pages 113–130.

[27] Tyler McDonnell, Baishakhi Ray, and Miryung Kim. An empirical study of api stability and adoption in the android ecosystem. In *Software Maintenance (ICSM), 2013 29th IEEE International Conference on*, pages 70–79. IEEE, 2013.

[28] Shinya Kasatani Patrick Lightbody Julian Harty Jennifer Bevan Haw-Bin Chai Philippe Hanrigou, Jason Huggins et al. selenium. `http://www.seleniumhq.org/`, 2008. [Online; accessed 19-July-2008].

[29] Hex-Rays SA. Ida pro. `https://www.hex-rays.com/products/ida/index.shtml`.

[30] N. Seriot. ios-runtime-headers. url = https://github.com/nst/iOS-Runtime-Headers. (Visited on 10/31/2015).

[31] Paulo de Barros SILVA FILHO. Static analysis of implicit control flow: resolving java reflection and android intents. 2016.

[32] Tielei Wang, Kangjie Lu, Long Lu, Simon Chung, and Wenke Lee. Jekyll on ios: When benign apps become evil. In *Presented as part of the 22nd USENIX Security Symposium (USENIX Security 13)*, pages 559–572, Washington, D.C., 2013. USENIX.

[33] Tim Werthmann, Ralf Hund, Lucas Davi, Ahmad-Reza Sadeghi, and Thorsten Holz. Psios: bring your own privacy & security to ios devices. In *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*, pages 13–24. ACM, 2013.

[34] Tom White. *Hadoop: The Definitive Guide, 3rd Edition.* O'Reilly Media / Yahoo Press, May 2012.

[35] Zhemin Yang, Min Yang, Yuan Zhang, Guofei Gu, Peng Ning, and X Sean Wang. Appintent: Analyzing sensitive data transmission in android for privacy leakage detection. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 1043–1054. ACM, 2013.

[36] Fang Yu, Yuan-Chieh Lee, Steven Tai, and Wei-Shao Tang. Appbeach: Characterizing app behaviors via static binary analysis. In *Proceedings of the 2013 IEEE Second International Conference on Mobile Services*, page 86. IEEE Computer Society, 2013.

[37] Jing Zhou and Robert J Walker. Api deprecation: a retrospective analysis and detection method for code examples on the web. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 266–277. ACM, 2016.

[38] Yajin Zhou, Zhi Wang, Wu Zhou, and Xuxian Jiang. Hey, you, get off of my market: detecting malicious apps in official and alternative android markets. In *NDSS*, volume 25, pages 50–52, 2012.