دانشگاه صنعتی شریف
دانشکده مهندسی کامپیوتر

پایان‌نامه کارشناسی ارشد
مهندسی رایانش امن

بهبود کارایی روش‌های تشخیص برنامک‌های اندرویدی بازبسته‌بندی شده

نگارش

مجتبی موذن

استاد راهنما

دکتر مرتضی امینی

اسفند ۱۴۰۱

به نام خدا

دانشگاه صنعتی شریف

دانشکده مهندسی کامپیوتر

پایان‌نامه کارشناسی ارشد

این پایان‌نامه به عنوان تحقق بخشی از شرایط دریافت درجه کارشناسی ارشد است.

عنوان: بهبود کارایی روش‌های تشخیص برنامک‌های اندرویدی بازبسته‌بندی شده

نگارش: مجتبی موذن

کمیته ممتحنین

استاد راهنما: دکتر مرتضی امینی امضاء:

استاد مشاور: استاد مشاور امضاء:

استاد مدعو: استاد ممتحن امضاء:

تاریخ:

سپاس

از استاد بزرگوارم که با کمک‌ها و راهنمایی‌های بی‌دریغشان، مرا در به سرانجام رساندن این پایان‌نامه یاری داده‌اند، تشکر و قدردانی می‌کنم. همچنین از پدر و مادر و خانواده‌ی عزیزم که مرا در اتمام این مسیر یاری رساندند، متشکر و قدردان آن‌ها هستم.

چکیده

با گسترش روزافزون استفاده از برنامک‌های اندرویدی در سالیان اخیر حملات موجود بر روی این سیستم‌عامل با افزایش قابل توجهی همراه بوده‌است. متن‌باز بودن برنامک‌های اندرویدی و در نتیجه، سهولت در دسترسی به کد منبع آن‌ها به جهت ایجاد تغییر، در کنار افزایش حملات بر روی آن‌ها ، لزوم توجه به مقابله با حملات مطروحه در این زمینه را افزایش داده‌است.حملات بازبسته‌بندی روی برنامک‌های اندرویدی، نوعی از حملات هستند که در آن مهاجم، پس از دسترسی به کد منبع برنامک و کپی‌کردن آن و یا ایجاد تغییراتی که مدنظر مهاجم است، مجدداً آن‌را بازبسته‌بندی می‌کند.تغییر کدهای برنامک، اهداف متفاوتی نظیر تغییر کتابخانه‌های تبلیغاتی، نقض امنیت کاربر و یا ضربه به شرکت‌های تولید برنامک از تغییر گسترش برنامک‌های جعلی را دنبال می‌کند. بازبسته‌بندی برنامک‌های اندرویدی علاوه بر ماهیت تهدید کاربران و شرکت‌ها، ماهیتی پیشگیرانه نیز دارد. در این حالت توسعه‌دهندگان نرم‌افزار با مبهم‌سازی در برنامک‌های اندرویدی، سعی در پیشگیری از بازبسته‌بندی به وسیله‌ی مهاجمان دارند. تشخیص بازبسته‌بندی در برنامک‌های اندرویدی از آن‌جهت دارای اهمیت است که هم کاربران و هم شرکت‌های توسعه‌دهنده، می‌توانند از این موضوع ذی‌نفع،باشند. تشخیص برنامک‌های بازبسته‌بندی شده، به جهت چالش‌های پیش‌رو، نظیر مبهم‌نگاری کدهای برنامک جعلی به دست مهاجم و همچنین تشخیص و جداسازی صحیح کدهای کتابخانه‌ای مسئله‌ای چالشی محسوب می‌شود. پژوهش‌های اخیر در این زمینه به صورت کلی، از روش‌های تشخیص مبتنی بر شباهت‌سنجی کدهای برنامک و یا طبقه‌بندی برنامک‌های موجود استفاده کرده‌اند. از طرفی برقراری حدواسطی میان سرعت و دقت در تشخیص برنامک‌های جعلی، چالشی است که استفاده از این دست پژوهش‌ها را در یک محیط صنعتی ناممکن ساخته‌است. در این پژوهش پس‌از استخراج کدهای برنامک به وسیله‌ی چارچوب سوت و ابزارهای دیس‌اسمبل، در یک روش دو مرحله‌ای کدهای برنامک‌های موجود با یکدیگر مقایسه می‌شود. پس از دیس‌اسمبل کدهای هر برنامک، در طی یک فرایند طبقه‌بندی مبتنی بر ویژگی‌های انتزاعی و دیداری، برنامک‌های کاندید برای هر برنامک مبدا استخراج می‌شود. سپس برای هر کلاس برنامک اندرویدی، امضایی متشکل از مهم‌ترین ویژگی‌های کدپایه از آن استخراج و پس از انجام مقایسه با کلاس‌های کتابخانه‌های اندرویدی موجود در مخزن، کتابخانه‌های اندرویدی حذف می‌شوند و در نهایت با مقایسه‌ی کدهای اصلی، برنامک بازبسته‌بندی شده مشخص می‌شود. در قسمت آزمون روش پیشنهادی در این پژوهش، توانستیم روش موجود در این زمینه‌را با بهبود امضای تولیدشده از هر برنامک و اضافه‌شدن مرحله‌ی پیش‌پردازش، سرعت تشخیص را ۶ برابر افزایش داده و در عین‌حال دقت روش موجود را نیز حفظ کنیم.

کلیدواژه‌ها: امنیت‌نرم‌افزار، اندروید، تشخیص‌بازبسته‌بندی، بهبودکارایی

فهرست مطالب

فهرست جدول‌ها

فهرست شکل‌ها

فصل ۱

مقدمه

سیستم‌عامل[۱] اندروید[۲] به دلیل سهولت در توسعه[۳] توسط توسعه‌دهندگان[۴] موبایلی و در نتیجه فراوانی استفاده از آن در تلفن‌های همراه، تلوزیون‌های هوشمند و دیگر دستگاه‌های موجود، حجم بالایی از بازار مصرفی سیستم‌عامل‌های موبایلی را به خود اختصاص داده‌است. بر طبق گزارش پایگاه استاتیستا[۵] [۶] سیستم‌عامل اندروید سهمی معادل ۷۱ درصدی از سیستم‌عامل‌های موبایلی را در سه ماهه‌ی پایانی سال ۲۰۲۲ به خود اختصاص داده‌است. در سال‌های اخیر به دلیل گسترش استفاده از این بستر[۶]، فروشگاه‌های اندرویدی زیادی به جهت ارائه‌ی خدمات به کاربران به وجود آمده‌است. برخی از فروشگاه‌های رسمی مانند فروشگاه اندرویدی گوگل[۷]، از ابزارهایی نظیر پلی پروتکت[۸] [۷] برای بررسی برنامک‌های اندرویدی موجود در فروشگاه استفاده می‌کنند. علاوه بر این، در سال‌های اخیر فروشگاه‌های متعدد رایگانی به وجود آمده‌اند که صرفا برنامک‌های اندرویدی موجود در سطح وب را جمع‌آوری و آن را به کاربران ارائه می‌دهند. فروشگاه‌های رایگان غالباً ابزارهای مشخصی را برای حفظ امنیت کاربران استفاده نمی‌کنند و امنیت کاربران این دسته از فروشگاه‌های اندرویدی، ، همواره تهدید می‌شود. یکی از راه‌های مورد استفاده توسط مهاجمان برای وارد ساختن بدافزار[۹] به تلفن‌های همراه، بازبسته‌بندی نرم‌افزار[۱۰] است. مطابق تعریف، بازبسته‌بندی شامل دانلود[۱۱] یک برنامک، دسترسی به محتوای کدهای برنامک اصلی از طریق

[۱] Operation System
[۲] Android
[۳] Development
[۴] Developers
[۵] Statista
[۶] Platform
[۷] Google
[۸] Play Protect
[۹] Malware
[۱۰] Software Repackaging
[۱۱] Download

روش‌های مهندسی معکوس[۱۲] و در نهایت بازبسته‌بندی به همراه تغییر و یا بدون تغییردادن کدهای برنامک اصلی[۱۳] است. زبان اصلی توسعه در برنامک‌های اندرویدی، زبان جاوا[۱۴] می‌باشد که یک زبان سطح بالا[۱۵] محسوب می‌شود. در طی فرآیند کامپایل[۱۶] برنامک‌های اندرویدی، مجموعه‌ی کدهای منبع در طی فرایندی به بایت‌کدهای دالویک[۱۷] تبدیل می‌شوند و در ادامه ماشین مجازی جاوا[۱۸]، بایت‌کدها را برروی ماشین مقصد اجرا می‌کند[۸]. فهم و در نتیجه مهندسی معکوس زبان میانی دالویک بایت‌کدها آسان است و به همین علت موجب سهولت در بازبسته‌بندی برنامک‌های اندرویدی می‌شود.

به طور کلی بازبسته‌بندی را می‌توان از دو جهت مورد بررسی قرار داد، از دید توسعه‌دهندگان، بازبسته‌بندی شامل فرآیندی است که توسعه‌دهنده با انجام مبهم‌نگاری[۱۹] در برنامک مورد توسعه، فهم بدنه‌ی اصلی برنامک را برای مهاجم[۲۰] سخت می‌کند. از این دید، بازبسته‌بندی یک روش تدافعی تلقی می‌شود تا مهاجم پس از دسترسی به کد برنامک اصلی، نتواند بدنه‌ی برنامک اصلی برنامک را شناسایی و در نتیجه آن‌را تغییر دهد. از جهت دیگر، بازبسته‌بندی توسط فردی که برنامک متعلق به او نیست یک عمل تهاجمی محسوب می‌شود. در این حالت، مهاجم پس از دسترسی به کد برنامک‌اصلی، بسته به هدف او، برنامک را مجددا بازبسته‌بندی می‌کند و آن را در فروشگاه‌های اندرویدی خصوصا فروشگاه‌هایی که نظارت کمتری بر روی آن‌ها وجود دارد منتشر می‌کند. در دیدگاه تهاجمی، مهاجم به جهت اهدافی متفاوتی نظیر تغییر کدهای تبلیغاتی[۲۱] در برنامک اصلی، تغییر درگاه‌های پرداخت و یا بازپخش بدافزار، اقدام به بازبسته‌بندی می‌کند. بازبسته‌بندی یکی از راه‌های محبوب مهاجمان برای انتقال بدافزارهای توسعه‌داده‌شده به تلفن همراه قربانی است[۹]. مطابق پژوهش آقای ژو و همکاران[۱۰] حدود ۸۵ درصد بدافزارهای موجود، از طریق بازبسته‌بندی منتشر می‌شوند. همانطور که گفته شد، برخی فروشگاه‌های اندرویدی نظیر گوگل، سازوکار مشخصی را برای تشخیص[۲۲] بازبسته‌بندی ارائه‌داده‌اند اما بسیاری از فروشگاه‌های اندرویدی فعال و پربازدید، خصوصا فروشگاه‌های رایگان، یا از هیچ ابزاری استفاده نمی‌کنند و یا در صورت توسعه‌ی نرم‌افزار بومی[۲۳] خود برای شناسایی برنامک‌های بازبسته‌بندی شده، مشخصات و یا دقت آن را گزارش نکرده‌اند.

همانطور که اشاره‌شد، به دلیل محبوبیت و در نهایت استفاده‌ی زیاد برنامک‌های اندرویدی و همچنین

[۱۲] Reverse Engineering
[۱۳] Orginal Application
[۱۴] Java
[۱۵] High Level
[۱۶] Compile
[۱۷] Dalvic Byte Code
[۱۸] Java Virtual Machine
[۱۹] Obfuscation
[۲۰] Attacker
[۲۱] Ad Code
[۲۲] Detection
[۲۳] Native

نظارت کم در فروشگاه‌های مرتبط، بازبسته‌بندی یک روش پر استفاده به جهت انتقال بدافزار به تلفن همراه کاربران است. آقای خانمحمدی و همکاران[۱۱]، پس از بررسی برنامک‌های اندرویدی مجموعه‌داده‌ی[۲۴] اندروزو[۲۵]، دریافتند که ٪۵۲/۲۲ از برنامک‌های موجود در این مخزن توسط ویروس‌توتال[۲۶]، بدافزار شناسایی شده‌اند. ویروس‌توتال، ابزاری متشکل از ۳۰ ضدبدافزار برای بررسی یک برنامک اندرویدی است. مطابق این پژوهش، ٪۷۷/۸۴ از برنامک‌های این مجموعه‌داده که بازبسته‌بندی شده‌اند، دارای نوعی از بدافزار ضدتبلیغاتی[۲۷] بوده‌اند که موجب می‌شود تبلیغات موجود در برنامک تغییر کرده و اهداف مالی و امنیتی کاربران و توسعه‌دهندگان مخدوش شود. علاوه بر این، مطابق پژوهشی که توسط ویداس وهمکاران[۱۲] انجام شده‌است، پس از پیاده‌سازی ۷ روش پربازدید به جهت تشخیص بازبسته‌بندی، در بهترین حالت، روش‌های موجود قادر به تشخیص ٪۷۲/۲۲ از برنامک‌های بازبسته‌بندی شده‌ی سه فروشگاه مطرح اندرویدی بوده‌اند. بنابراین مشخص است که تشخیص برنامک‌های بازبسته‌بندی شده، به چه میزان می‌تواند اهداف مالی و امنیتی توسعه‌دهندگان و کاربران برنامک‌ها را ارضا کند. در سال‌های اخیر ارائه‌ی یک راهکار پرسرعت به همراه دقت مناسب، همواره یکی از دغدغه‌های مهم پژوهش‌گران در این زمینه بوده‌است.

همانطور که گفته‌شد، بازبسته‌بندی برنامک‌های اندرویدی از دو دیدگاه تهاجمی و تدافعی قابل بررسی است. در حالتی که کاربر متقلب، برنامک اندرویدی اصلی را دچار تغییراتی می‌کند و آن را در اختیار عموم قرار می‌دهد، تشخیص بازبسته‌بندی، با استفاده از مقایسه‌ی برنامک اصلی و برنامک جعلی صورت می‌گیرد. تشخیص بازبسته‌بندی در این حالت را می‌توان به صورت کلی به دو طبقه تقسیم کرد. در حالت اول توسعه‌دهنده روش خود را مبتنی بر تحلیل برنامک مبدا و مقصد پیاده‌سازی می‌کند. عمده‌ی روش‌های موجود در این طبقه مبتنی بر تحلیل ایستا[۲۸]ی جفت برنامک‌ها است و استفاده از تحلیل پویا[۲۹] به جهت سرعت پایین آن، محبوبیت فراوانی ندارد[۱۳]. در سمت دیگر طبقه‌بندی[۳۰] برنامک‌های اندرویدی وجود دارد.روش‌های موجود در این دسته، عمدتا سرعت بالایی دارند اما در تشخیص جفت بازبسته‌بندی شده دقت پایینی را ارائه می‌دهند.

برنامک‌های اندرویدی متشکل از دو قسمت اصلی کدهای برنامک و منابع[۳۱] هستند. کدهای برنامک، منطق[۳۲] برنامک را تشکیل می‌دهند و رفتار برنامک با توجه به این قسمت مشخص می‌شود. از طرفی منابع

[۲۴] Data Set
[۲۵] Androzoo
[۲۶] Virus total
[۲۷] AdWare
[۲۸] Static
[۲۹] Dynamic
[۳۰] Classification
[۳۱] Resouurces
[۳۲] Logic

برنامک، رابط کاربری[۳۳] آن را تشکیل می‌دهند. روش‌های مبتنی بر تحلیل برنامک و یا طبقه‌بندی آن، عمدتاً از ویژگی‌های موجود در منابع و یا کد آن استفاده می‌کنند. مهاجم در حالاتی که می‌خواهد از محبوبیت برنامک مبدا استفاده کند، سعی در یکسان‌سازی ظاهر برنامک‌های مبدا و مقصد دارد به همین جهت از منابع برنامک مبدا استفاده می‌کند و منطق برنامک را مطابق با اهداف خود تغییر می‌دهد. در حالتی دیگر، متقلب سعی می‌کند که با استفاده از تغییر منابع برنامک و تولید یک برنامک تقلبی و گاهاً بدون هیچ تغییری در کد برنامک، ادعای توسعه‌ی یک برنامک جدید را اثبات کند. لازم به ذکر است استفاده از ویژگی‌های کدپایه[۳۴] و منبع‌پایه[۳۵]، به وفور در پژوهش‌های سال‌های اخیر یافت می‌شود که هر کدام معایب و مزایای خود را دارد.

در روش‌های مبتنی بر طبقه‌بندی عمدتا تعریف تشخیص بازبسته‌بندی محدود به تشخیص دسته‌ی مشکوک که احتمال بازبسته‌بندی بودن جفت‌های داخل این دسته، بیش از سایر دسته‌ها است. تشخیص بازبسته‌بندی در این روش‌ها، محدود به تشخیص طبقه‌ی برنامک ورودی می‌باشد و جفت بازبسته‌بندی شده مشخص نمی‌شود. از طرفی در روش‌های مبتنی بر تحلیل برنامک، بررسی دوبه‌دوی برنامک‌های ورودی و مجموعه‌داده مدنظر است. در این روش‌ها تعریف تشخیص بازبسته‌بندی گسترش یافته و یافتن جفت بازبسته‌بندی به صورت مشخص، از اهداف اصلی پژوهش است. تغییر منابع برنامک و همچنین مبهم‌نگاری در برنامک بازبسته‌بندی شده، دو چالش مهم در راستای تشخیص بازبسته‌بندی است. متقلب پس از بازبسته‌بندی برنامک، با استفاده از مبهم نگاری سعی می‌کند تغییرات خود و شباهت ساختار منطقی برنامک تقلبی با برنامک اصلی را پنهان کند. به همین جهت، تشخیص بازبسته‌بندی نیازمند ویژگی‌هایی است که مقاومت بالایی مقابل مبهم‌نگاری داشته‌باشد بدین معنا که تغییر و ایجاد ابهام در کد، به راحتی در این ویژگی‌ها قابل انجام نباشد.

در هنگام کامپایل برنامک‌های اندرویدی، کتابخانه‌ها[۳۶]یی که در برنامک مورد استفاده قرار گرفته‌اند به همراه کد مورد توسعه، کامپایل شده و بایت‌کدهای دالویک آن در کنار برنامک قرار می‌گیرد. بر اساس پژوهش آقای زیانگ و همکاران [۱۴] ٪۵۷ از کدهای برنامک‌های مورد بررسی در این پژوهش، شامل کدهای کتابخانه‌ای بودند که دچار مبهم‌نگاری نشده‌اند. بنابراین تشخیص کدهای بازبسته‌بندی شده بدون تشخیص درست و دقیق و جداسازی کدهای کتابخانه‌ای امکان‌پذیر نیست و در صورتی که به درستی جداسازی صورت گیرد، می‌تواند نتایج منفی غلط و مثبت غلط را کاهش دهد. به صورت کلی دو روش برای تشخیص کدهای کتابخانه‌ای استفاده می‌شود، روش مبتنی بر لیست سفید[۳۷] و روش تشخیص مبتنی بر

[۳۳] User Interface
[۳۴] Code Base
[۳۵] Resource Base
[۳۶] Library
[۳۷] White List

شباهت سنجی[۳۸]. در روش لیست سفید، لیستی از مشهورترین کتابخانه‌های موجود در مخازن کتابخانه‌ای اندروید نظیر ماون[۳۹] جمع آوری می‌شود و با استفاده از نام کلاس‌ها و بسته‌های موجود، کلاس‌های کتابخانه ای تشخیص‌داده می‌شود. مشخص است که این روش، مقاومت بسیار کمی مقابل ساده‌ترین روش‌های مبهم‌نگاری در کتابخانه‌های اندرویدی دارد. در حالت دیگر از روش‌های مبتنی بر شباهت سنجی برای تشخیص کدهای کتابخانه‌ای استفاده می‌شود که در این روش، تحلیل ایستا روی کدهای برنامک مبدا و مخزن کتابخانه‌های اندروید صورت می‌گیرد و در نهایت از طریق شباهت‌سنجی، کدهای کتابخانه‌ای تشخیص داده می‌شوند. مشخص است که روش‌های مبتنی بر شباهت‌سنجی از دقت بیشتری، خصوصا در صورت وجود ابهام، برخوردار هستند و تمایز بهتری میان کدهای کتابخانه‌ای و کدهای اصلی قرار می‌دهند اما اینگونه روش‌ها سرعت پایینی دارند.

پژوهش‌های ارائه‌شده در زمینه‌ی تشخیص برنامک‌های بازبسته‌بندی شده در سال‌های اخیر، عمدتاً در تلاش برای بهبود دقت و سرعت روش‌های پیشین بوده‌اند.مبهم‌نگاری باعث می‌شود که دقت روش‌های تشخیص مبتنی بر تحلیل ایستا و شباهت سنجی پایین بیاید و لزوم استفاده از ویژگی‌هایی را که مقاومت بالایی مقابل مبهم‌نگاری داشته باشند را افزایش دهد. از طرفی استفاده از ویژگی‌های مقاوم به مبهم‌نگاری، می‌تواند سرعت تشخیص را بسیار پایین آورده تا حدی که عملا استفاده از این روش‌ها در یک محیط صنعتی را غیر ممکن سازد. در این پژوهش ما با استفاده از ترکیب روش‌های تحلیل ایستا و طبقه‌بندی منابع، به همراه شباهت‌سنجی، روشی را ارائه کرده‌ایم که در حالی که مقاومت بالایی نسبت به مبهم‌نگاری داشته‌باشد، سرعت روش‌های پیشین را نیز افزایش دهد. در این پژوهش به عنوان پیش‌پردازش، از یک طبقه‌بند[۴۰] نزدیک‌ترین همسایه[۴۱] برای کاهش فضای مقایسه‌ی دودویی[۴۲] و با استفاده از ویژگی‌های مبتنی بر منبع، سرعت تشخیص بهبود داده‌شده است. با کاهش فضای مقایسه‌ی دودویی و طبقه‌بندی برنامک‌های مشکوک در یک دسته، مقایسه‌ی برنامک‌های موجود در آن دسته آغاز می‌شود. مقایسه‌ی دودویی در هر دسته مبتنی بر تحلیل ایستا و شباهت‌سنجی کدهای برنامک انجام می‌شود. ابتدا ویژگی‌هایی از هر کلاس و متد[۴۳] در بسته‌های برنامک استخراج شده و امضا[۴۴]ی هر کلاس ساخته می‌شود به طوری که امضای هر کلاس منحصر به فرد و تا حد امکان مختص همان کلاس باشد. نوآوری روش مطروحه، ترکیب روش‌های مبتنی بر طبقه‌بندی و روش‌های مبتنی بر تحلیل ایستا می‌باشد که در نهایت منجر به افزایش سرعت و در عین حال دقت خوب در تشخیص برنامک‌های بازبسته‌بندی شده‌است. حذف کدهای کتابخانه‌ای با استفاده از روشی مبتنی بر مقایسه‌ی کدهای موجود در مخزن کتابخانه‌ها و کلاس‌های برنامک انجام می‌شود. مخزن

[۳۸] Similarity
[۳۹] Maven Repository
[۴۰] Classifier
[۴۱] Nearest Neighbor
[۴۲] Pairwise Comparison
[۴۳] Method
[۴۴] Signature

کتابخانه‌ها متشکل از ۸۷۷ کتابخانه‌ی اندرویدی جمع‌آوری شده از مخزن ماون[۴۵] می‌باشد. در نهایت پس از تشخیص کلاس‌های کتابخانه‌های اندرویدی و حذف آن‌ها از کد برنامک، کدهای مورد توسعه به عنوان ورودی برای مقایسه‌ی دودویی و طبقه‌بندی مورد تحلیل قرار می‌گیرند. بهبود امضای کلاسی و در نهایت تولید امضای برنامک‌های اندرویدی با استفاده از ویژگی‌های مقاوم و در عین حال کوتاه و همچنین استفاده از روشی مبتنی بر طبقه‌بندی برنامک‌های اندرویدی پیش از مقایسه‌ی دودویی آن‌ها، ایده‌ی اصلی این پژوهش برای تشخیص برنامک‌های اندرویدی بازبسته‌بندی شده بوده‌است.

در ادامه‌ی این نگارش، در فصل ۲ به تعریف مفاهیم اولیه مورد نیاز این پژوهش می‌پردازیم. در فصل ۳ به تعریف مسئله می‌پردازیم و همچنین مروری از کارهای پیشین را خواهیم داشت. در ادامه و در فصل ۴ روش مورد استفاده در این پژوهش، شرح داده خواهدشد و در فصل ۵ مقایسه و ارزیابی روش پیشنهادی خود را ارائه می‌دهیم. در نهایت و در فصل ۶ ضمن جمع‌بندی این گزارش علمی، به بررسی نقاط ضعف و قوت این پژوهش و همچنین ارائه‌ی پیشنهاداتی جهت بهبود آن خواهیم‌پرداخت.

[۴۵] Maven Repository

فصل ۲

مفاهیم اولیه

در این فصل مفاهیمی را که به صورت مستقیم و غیرمستقیم در این پژوهش از آن‌ها استفاده شده‌است را شرح می‌دهیم. آشنایی با مفاهیم مطروحه در این فصل، منجر به درک بهتر پژوهش و راه‌حل پیشنهادی در فصل ۴ خواهد شد.

۲ـ۱ مبهم‌سازی

آن‌چنان که در فصل پیشین گفته‌شد، مبهم‌سازی را می‌توان از دو دیدگاه تهاجمی و تدافعی بررسی کرد. در این قسمت ما با توجه به هدف پژوهش که تشخیص بازبسته‌بندی به جهت دفاع می‌باشد، مبهم‌سازی را فرایندی در نظر می‌گیریم که در آن فرد مهاجم یا به بیان دیگر متقلب، برنامک اصلی را دانلود کرده و پس از دیکامپایل[۱] کردن، به نوعی تغییر می‌دهد که منطق کلی برنامک، تغییری نمی‌کند. مبهم‌سازی یکی از ارکان اصلی در فرایند بازبسته‌بندی است و هدف اصلی آن این است که ابزارهای تشخیص بازبسته‌بندی، خصوصا در مواردی که از تحلیل ایستا استفاده می‌کنند را به اشتباه بیاندازد.

روش‌های مبهم‌سازی را از نظر میزان سختی در تشخیص به سه دسته‌ی کلی می‌توان تقسیم کرد[۱۵]:

۲ـ۱ـ۱ روش‌های ساده

راهکارهای موجود در این دسته عمدتاً بدون تغییر در برنامک اصلی رخ می‌دهد. در این روش متقلب پس از آن‌که به کدهای برنامک اصلی دسترسی پیدا کرد، آن را بدون هیچ گونه تغییری کامپایل و بسته‌بندی می‌کند.

[۱] Decompile

بازبسته‌بندی تنها موجب تغییر در امضاء توسعه‌دهنده‌ی برنامک و جمع‌آزما[۲] می‌شود. بنابراین روش‌هایی که مبتنی بر این دو خصوصیت هستند در این سطح دچار مشکل می‌شوند.

۲ـ۱ـ۲ روش‌های میانی

این دسته از روش‌های مبهم‌سازی، شامل روش‌هایی است که در آن بیشتر ویژگی‌های مبتنی بر معناشناسی[۳] تغییر می‌کند و ویژگی‌های مبتنی بر نحو[۴] ثابت باقی می‌مانند. بنابراین، روش‌هایی که بیشتر مبتنی بر معناشناسی برنامک‌های اندرویدی هستند، دچار خطای بیشتری در این سطح از مبهم‌نگاری می‌شوند. در ادامه به معرفی مختصری از انواع روش‌های مبهم‌نگاری مطابق با پژوهش [۱۶] در این دسته می‌پردازیم:

- **تغییر نام شناسه[۵]ها:** تغییر نام شناسه‌های موجود در برنامک شامل نام کلاس‌ها، متدها و یا متغیرها[۶]ی موجود [۱۵]

```
public class a{
    private Integer a;
    private Float = b;
    public void a(Integer a, Float b){
        this.a = a + Integer.valueOf(b)
    }
}
```

شکل ۲ـ۱: نمونه‌ای از مبهم‌نگاری با استفاده از تغییر نام شناسه‌ها

- **تغییر نام بسته:** در این روش مبهم‌نگاری با استفاده از تغییر نام بسته‌های برنامک صورت می‌گیرد.
- **رمزنگاری رشته‌ها:** استفاده از رمزنگاری در رشته‌های[۷] مورد استفاده در فایل‌های دکس[۸]،باعث کاهش سطح معناشناسی[۹] و در نهایت مبهم‌شدن رشته‌ها می‌شود.
- **فراخوانی غیرمستقیم:** یکی از روش‌های ساده‌ی تغییر گراف فراخوانی[۱۰]، استفاده از یک تابع واسط به عنوان تابع فراخواننده‌ی[۱۱] تابع اصلی است. در این حالت تابع اولیه یک تابع واسط و تابع واسط

[۲] Checksum
[۳] Semantic
[۴] Syntax
[۵] Identifier
[۶] Variable
[۷] String
[۸] Dex Files
[۹] Semantic Level
[۱۰] Call Graph
[۱۱] Caller

به صورت زنجیره‌ای تابع اصلی را فراخوانی می‌کند. بدنه‌ی تابع واسط در این حالت، بسیار ساده و شامل یک فراخوانی تابع اصلی است.

- **جابه‌جایی دستورات:** جابه‌حایی دستورات موجود در برنامک اصلی، یکی از روش‌های پرکاربرد توسط ایزارهای مبهم‌نگاری است. جابه‌جایی دستورات به شکلی انجام می‌شود که استقلال هر قسمت حفظ گردد.

- **جابه‌جایی ساختار سلسله‌مراتبی:** در این روش، ساختار سلسله مراتبی کلاس‌های برنامک به نوعی تغییر می‌کتد که منطق کلاس‌ها دچار تغییر نشود.

- **ادغام و شکستن:** می‌توان توابع و یا کلاس‌های موجود در برنامک‌های اندرویدی را ادغام کرد. برای مثال می‌توان هر جایی که یک تابع صدا زده‌شده بود، فراخوانی تابع با بدنه‌ی تابع جایگزین شود. از طرفی می‌توان بدنه‌ی چند تابع را تحت یک تابع با یکدیگر ادغام کرد. این کار ساختار توابع فراخواننده را نیز تغییر می‌دهد. از طرفی می‌توان یک تابع را به چندین تابع مشخص شکست و بدین صورت گراف جریان برنامک را تغییر داد.

- **واردساختن کدهای بیهوده:** کدهای بیهوده، کدهایی هستند که اجرا می‌شوند ولی تاثیری در ادامه‌ی روند اجرایی برنامک، ندارند. کدهای بیهوده عموماً دارای ساختارهای کنترلی[۱۲] و حلقه‌های نُپ[۱۳] هستند که تاثیری در روند اجرای برنامک ندارند. ذکر این نکته حائز اهمیت است که در صورتی که در ساختار کدهای بیهوده از شروط کنترلی مبتنی بر متغیرهای پویا[۱۴] استفاده شود آنگاه دیگر تحلیل ایستای برنامک‌های اندرویدی قادر به تشخیص این نوع از مبهم‌نگاری‌ها نیست.

- **واردساختن کدهای مرده:** یکی دیگر از روش‌های تغییر گراف‌های برنامه از جمله گراف جریان[۱۵]، اضافه‌کردن کدهای مرده‌ای است که در ساختار گراف جریان برنامک‌های اندرویدی هیچ‌گاه اجرا نمی‌شوند اما به عنوان یک گره در گراف حضور دارند.

- **روش‌های دیگر:** روش‌های دیگری نظیر تغییر نام منابع مورد استفاده در برنامک‌های اندرویدی و حذف فایل اشکال‌زدایی[۱۶] از روش‌های دیگری است که در این سطح به وفور مورد استفاده قرار می‌گیرد.

---

[۱۲] Control's Statement
[۱۳] Nop
[۱۴] Dynamic Variables
[۱۵] Flow Graph
[۱۶] Debug File

۲ـ۱ـ۳ روش‌های خاکستری

روش‌های موجود در این دسته، مبتنی بر نحوِ برنامک‌های اندرویدی و خصوصا زبان جاوا به وجود آمده‌است. عمده‌ی روش‌های مورد استفاده در این سطح، از خصوصیات مهم زبان جاوا به عنوان زبان اصلی در پیاده‌سازی برنامک‌های اندرویدی، استفاده می‌کنند. در ادامه به بررسی مهم‌ترین روش‌های موجود در این دسته می‌پردازیم.

- **بازتاب**[۱۷]: بازتاب یکی از ویژگی‌های مهم و پیچیده‌ی زبان جاوا می‌باشد[۱۷] که امکان فراخوانی متدها و ارتباط با کلاس‌های برنامک را به صورت پویا فراهم می‌سازد. مهاجمان با استفاده از فراخوانی متدها به وسیله‌ی قابلیت بازتاب، می‌توانند نام واسط فراخوانی‌شده را پنهان سازند و بدین وسیله سطح جدیدی از مبهم‌نگاری را در برنامک‌های اندرویدی ایجاد سازند. استفاده از قابلیت بازتاب و رمزنگاری[۱۸] رشته‌ی واسط مورد نظر، به طور کامل واسط فراخوانی‌شده را پنهان می‌کند.

```
1 Object object = new Object();
2 Method getService = Class.forName("android.os.
      ServiceManager").getMethod("getService",
      String.class);
3 Object obj = getService.invoke(object, new
      Object[]{new String("batteryinfo")});
```

شکل ۲ـ۲: نمونه‌ای از مبهم‌نگاری با استفاده از قابلیت بازتاب به منظور پنهان‌سازی واسط فراخوانی‌شده به نام batteryinfo

- **رمزنگاری دالویک بایت‌کدها:** در این روش، مهاجم در حین ساختن برنامک بازبسته‌بندی شده، قسمتی مهمی از کدهای برنامک را رمزنگاری کرده و در هنگام اجرا با استفاده از یک رویه‌ی رمزگشایی[۱۹]، کدهای اصلی را بارگیری[۲۰] می‌کند. این روش عمدتاً زمانی استفاده می‌شود که مهاجم نیاز به فراخوانی توابع واسط‌های برنامه‌نویسی داشته‌باشد و قسمتی را که واسط‌ها فراخوانی می‌شوند را رمزنگاری می‌کند.

- **بارگذاری پویای کلاس‌ها**[۲۱]: زبان جاوا از قابلیت مهمی به نام بارگیری پویای کد پشتیانی می‌کند که اجازه می‌دهد تکه کدی را که پیش از این در کد مورد توسعه‌ی یک برنامک موجود نبوده را در حین اجرا به برنامک اضافه کنیم. مهاجم با استفاده از این قابلیت زبان جاوا می‌تواند در حین اجرای

[۱۷] Reflect
[۱۸] Encryption
[۱۹] Decryption
[۲۰] Load
[۲۱] Dynamic Class Loading

برنامک، قسمت‌هایی را به برنامه اضافه‌کند که عملاً تشخیص آن‌ها با استفاده از تحلیل‌های ایستا امکان‌پذیر نیست.

۲ـ۱ـ۴ روش‌های ترکیبی

هر ترکیبی از روش‌های گفته‌شده در سطوح مختلف را می‌توان برای مبهم‌نگاری استفاده کرد. به صورت کلی روش‌های میانی ۲ـ۱ـ۲ و روش‌های خاکستری ۲ـ۱ـ۳ را می‌توان دو دسته‌ی مهم از انواع مبهم‌نگاری به حساب آورد که به صورت گسترده در مبهم‌نگارهای رایگان و یا تجاری مورد استفاده قرار می‌گیرد.

۲ـ۱ـ۵ انواع مبهم‌نگارها

در قسمت پیشین، دریافتیم که مبهم‌نگاری، سطوح متفاوتی دارد که متقلبان برای تولید برنامک‌های بازبسته‌بندی شده از آن‌ها استفاده می‌کنند.از آن‌جایی که بسیاری از برنامک‌های تقلبی با استفاده از مبهم‌نگار[۲۲]ها توسعه یافته‌اند و علاوه بر این برای ابداع یک روش مفید جهت تشخیص برنامک‌های بازبسته‌بندی شده ابتدا باید انوع مبهم‌نگارهای موجود را بررسی کرد. در پژوهشی که توسط ژانگ و همکاران[۱۸] انجام شده، ٪۴۳ از برنامک‌های بازبسته‌بندی شده‌ی مورد بررسی در این پژوهش، از مبهم‌نگاری‌های بسیار ساده‌ای نظیر تغییر نام و با استفاده از مبهم‌نگارهای رایگان، انجام شده‌است. در ادام به بررسی چند مبهم‌نگار رایگان و تجاری[۲۳] می‌پردازیم.

- **پروگارد**

پروگارد[۲۴] یک نرم‌افزار متن‌باز رایگان به جهت بهینه‌سازی و مبهم‌نگاری در برنامه‌های جاوا مورد استفاده قرار می‌گیرد. بهینه‌سازی از طریق حذف کدهای مرده و منابع بلااستفاده انجام می‌شود و مبهم‌نگاری عمدتاً با استفاده روش‌های مشروحه در بخش ۲ـ۱ـ۳ انجام می‌شود.[۱۹]

- **آلاتوری**

آلاتوری[۲۵] یک مبهم‌نگار رایگان تولیدشده توسط شرکت روسی اسماردک[۲۶] می‌باشد که سطوح مختلفی از مبهم‌نگاری را با توجه به فایل‌های پیکربندی[۲۷] پوشش می‌دهد. این مبهم‌نگار از تغییرنام،

[۲۲] Obfuscator
[۲۳] Commercial
[۲۴] Proguard
[۲۵] Allatori
[۲۶] Smardec
[۲۷] Configuration

مبهم‌نگاری مبتنی بر تغییر گراف‌های جریان، مبهم‌نگاری فایل‌های اشکال‌زدایی و رمزنگاری داده‌های رشته‌ای پشتیبانی می‌کند.[۲۰، ۲۱]

- **دکس‌گارد**

این مبهم‌نگار نسخه‌ی تجاری نرم‌افزار پروگارد است که توسط شرکت گارداسکوار[۲۸] تولید شده‌است. دکس‌گارد[۲۹] را می‌توان مشهورترین و یکی از پیچیده‌ترین مبهم‌نگارهای موجود به حساب آورد. آخرین نسخه‌ی این نرم‌افزار انواع مبهم‌نگاری‌های سطح خاکستری نظیر بارگیری پویای کد و همچنین رمزنگاری کلاس‌ها و توابع را به صورت کامل انجام می‌دهد.

۲ـ۲ ساختار فایل‌های برنامک‌های اندرویدی

هر برنامک اندرویدی یک فایل فشرده‌شده با پسوند apk[۳۰] است که به اختصار شامل چهار پوشه‌ی مهم و سه فایل می‌باشد.برای درک بهتر از روش پیشنهادی این پژوهش، در ادامه هر کدام از این قسمت‌ها را معرفی و کارکرد آن را بررسی خواهیم‌کرد[۲۲]. شمای کلی از ساختار برنامک‌های اندرویدی را می‌توان در شکل ۲ـ۳ مشاهده نمود.

- **پوشه‌ی res** :این پوشه شامل منابع برنامک‌های اندرویدی است که مربوط به رابط کاربری برنامک می‌شود. این پوشه در نهایت به فایل‌های R. نگاشت شده و هر کدام از منابع با یک شناسه[۳۱] مشخص می‌گردد.

- **پوشه‌ی lib:** فایل‌های کامپایل‌شده‌ی بومی در این پوشه قرار می‌گیرند که شامل کتابخانه‌های اندرویدی و جاوایی نیز می‌شود. استفاده از فایل‌هایی که کامپایل شده‌اند سرعت اجرای برنامک‌های اندرویدی را بالا می‌برد لذا استفاده از آن‌ها به عنوان بسته‌های[۳۲] از پیش آماده محبوبیت دارد.

- **فایل Classes.dex:** فایل‌های با پسوند dex فایل‌های دودویی[۳۳] هستند که اطلاعات را در سطر و ستون‌های خود ذخیره می‌کنند. این فایل در برنامک‌های اندرویدی حاوی بایت‌کدهای دالویک است که توسط ماشین مجازی دالویک[۳۴] اجرا می‌شود.

---

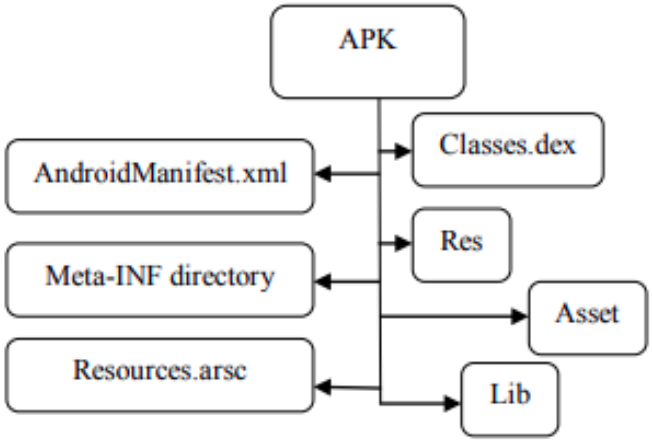
[۲۸] Guardsquare
[۲۹] DexGaurd
[۳۰] Android Package
[۳۱] Resource Id
[۳۲] Module
[۳۳] Binary
[۳۴] Dalvik Virtual Machine

- **فایل AndroidManifest.xml:** پیکربندی‌های مهم فایل‌های apk از جمله لیست مجوزهای مورد نیاز، لیست مولفه‌ها[۳۵] و نام بسته‌ی برنامک در این فایل نوشته می‌شود.

- **پوشه‌ی assets:** این پوشه همانند پوشه‌ی res برای منابع ایستا مورد استفاده قرار می‌گیرد با این تفاوت همه توسعه‌دهندگان در این پوشه می‌توانند عمق زیرپوشه‌ها را به تعداد نامتناهی افزایش دهند تا ساختار بهتری را فراهم سازند.

- **پوشه‌ی META-INF:** این پوشه شامل اطلاعات کلیدهای عمومی[۳۶] کاربر توسعه‌دهنده‌ی برنامک است که برنامک با کلید خصوصی متناظر آن امضا شده‌است. امضای موجود در این پوشه، خاصیت صحت‌سنجی[۳۷] دارد اما اطلاعاتی را از توسعه‌دهنده نشر نمی‌دهد و به صورت خودامضا ساخته می‌شود.

- **فایل resources.arsc:** این فایل برای انجام نگاشت[۳۸] میان منابع موجود در پوشه‌ی $resources$ و شناسه‌ی هر منبع استفاده می‌شود تا بتوان در حین اجرای برنامک‌ها، هر شناسه را به منبع آن ترجمه کرد.



شکل ۲ـ۳: ساختار پوشه‌ها و فایل‌های بسته‌های apk [۲]

---

[۳۵] Component
[۳۶] Public Key
[۳۷] Integrity
[۳۸] Mapping

۲ـ۳ کتاب‌خانه‌های اندرویدی

کتابخانه‌های اندرویدی ، بسته‌های از پیش توسعه‌یافته هستند که توسط توسعه‌دهندگان نوشته شده و توسعه‌دهندگان اندروید به جهت سهولت در پیاده‌سازی و کمک به تسریع توسعه‌ی نرم‌افزار به وفور از این نمونه‌ها استفاده می‌کنند. کتابخانه‌های اندرویدی به صورت کلی به دوبخش کتابخانه‌های مختص برنامه‌نویسی اندرویدی و کتاب‌خانه‌های زبان جاوا تقسیم می‌شوند. در هنگام کامپایل، تمامی کتاب‌خانه‌هایی که توسعه‌دهنده هنگام توسعه‌ی برنامک، آن‌ها را استفاده کرده‌است به همراه کدهای مورد توسعه، کامپایل شده و در ساختار سلسله مراتبی تحت فایل‌های classes.dex قرار می‌گیرد[۲۳]. ذکر این نکته قابل توجه است که تشخیص برنامک‌های بازبسته‌بندی شده بدون شناسایی کتابخانه‌های برنامک اندرویدی مورد نظر امکان‌پذیر نیست. واضح است که در صورتی که نتوانیم کتاب‌خانه‌های اندرویدی شاخص را از جفت برنامک‌های مورد بررسی جدا کنیم، آن‌گاه بخش زیادی از شباهت دو برنامک ناشی از کتاب‌خانه‌های اندرویدی و اشتراکات موجود در آن‌ها است چرا که بسیاری از کتابخانه‌ها خصوصا کتاب‌خانه‌های زبان جاوا، در هر برنامک اندرویدی موجود است. از طرفی، به دلیل عدم وجود مرز مشخصی میان کدهای کتابخانه‌ای و کدهای مورد توسعه توسط توسعه‌دهندگان، شناسایی کتابخانه‌های اندرویدی تبدیل به یک چالش در زمینه‌ی تشخیص برنامک‌های بازبسته‌بندی در این حوزه شده‌است.

۲ـ۴ طبقه‌بندی

طبقه‌بندی اطلاعات ورودی یکی از روش‌های مرسوم در هوش‌مصنوعی و یادگیری ماشین است که توسط الگوریتم‌های طبقه‌بند انجام می‌شود. یک طبقه‌بند شامل مجموعه‌ای از الگوریتم[۳۹]ها است که برای طبقه‌بندی و یا مرتب‌سازی[۴۰] داده‌های ورودی مورد استفاده قرار می‌گیرد [۲۴]. یکی از ساده‌ترین مثال‌های موجود برای طبقه‌بندی، جداسازی هرزنامه[۴۱]ها در سرویس‌های ایمیل است.روش‌های طبقه‌بندی نیازمند مجموعه‌ای از ویژگی‌های اطلاعات مورد بررسی به عنوان ورودی مسئله می‌باشند تا پس از اجرای الگوریتم، اطلاعات مسئله را بر اساس آن‌ها طبقه‌بندی کنند.

[۳۹] Algorithm
[۴۰] Sorting
[۴۱] Spam

۲ـ۵ بازبسته‌بندی برنامک‌های اندرویدی

با پویش عمیق در پژوهش‌های مرتبط با این حوزه در سالیان اخیر متوجه می‌شویم که تعاریف متنوعی برای بازبسته‌بندی در نظر گرفته‌شده است. برخی از پژوهش‌ها نظیر [۲۵، ۲۶] بازبسته‌بندی را در تغییر منابع و ظاهر برنامک‌ها در نظر می‌گیرند و در نهایت ویژگی‌های مبتنی بر ظاهر آن‌ها را با یکدیگر مقایسه می‌کنند. در حالی که برخی از پژوهش‌های اخیر نظیر [۲۷، ۲۸] بازبسته‌بندی را مبتنی بر تغییر ویژگی‌های کدپایه تعریف کرده‌اند.البته که نمی‌توان به هیچ کدام از تعاریف بالا خرده گرفت چرا که هر دو تعریف از نظر مهاجم و اهداف تعریف شده توسط او قابل استناد است. علاوه بر این یکی دیگر از اختلافات موجود در تعریف بازبسته‌بندی، وجود مبهم‌نگاری در برنامک‌های بازبسته‌بندی شده‌است. برخی از پژوهش‌ها نظیر [۲۹] بازبسته‌بندی را منوط به تغییر در امضای برنامک می‌دانند اما بسیاری از پژوهش‌های به‌روزتر، نظیر [۳۰، ۳۱] بازبسته‌بندی را تنها به تغییر منابع و یا کدهای برنامک تقلبی نسبت به برنامک اصلی می‌دانند. همانطور که مشاهده شد، هنوز تعریف مشخصی از بازبسته‌بندی در پژوهش‌ها ارائه نشده‌است اما به طور کلی می‌توان بازبسته‌بندی را به صورت زیر تعریف کرد:

تعریف ۲ـ۱ (بازبسته‌بندی) *برنامک A بازبسته‌بندی یک برنامک دیگر است اگر تغییرات آن نسبت به برنامک مبدا محدود و با حفظ کارکرد و منابع برنامک اصلی باشد.*

این تعریف در این پژوهش نیز به عنوان تعریف مبنای بازبسته‌بندی در نظر گرفته شده‌است.

فصل ۳

تعریف مسأله و مرور کارهای پیشین

پژوهش‌های اخیر در حوزه‌ی تشخیص برنامک‌های اندرویدی بازبسته‌بندی شده نشان می‌دهد که تشخیص این دسته از برنامک ها تحت تاثیر دو عامل مبهم‌نگاری و جداسازی صحیح کتابخانه‌های اندرویدی قرار دارد. برخی از پژوهش‌های اخیر انجام‌شده در این حوزه، تشخیص کتاب‌خانه‌های بسته‌ی تقلبی را با فرض عدم مبهم‌نگاری کتابخانه‌ها انجام داده‌اند که مشخصاً این فرضی نادرست است چرا که بسیاری از مبهم‌نگارهای ابتدایی نیز این کار را در کتابخانه‌های اندرویدی انجام می‌دهند. در اکثر روش‌های پیشنهادی قسمتی از روش، مختص تشخیص و جداسازی کتابخانه‌های اندرویدی است. شناسایی کدهای کتابخانه‌ای از آن‌جهت اهمیت دارد که تشخیص درست آن‌ها می‌تواند خطای مثبت غلط و منفی غلط را کاهش دهد. در بیشتر مواقع، خصوصا در ابزارهای مبهم‌نگاری، متقلب هنگام بازبسته‌بندی اقدام به مبهم‌نگاری در کتابخانه‌های اندرویدی می‌کند و بدین صورت سعی در افزایش منفی غلط در ابزارهای تشخیص دارد. در صورتی که کدهای کتابخانه‌ای به درستی تشخیص و جداسازی نشوند، شباهت‌های موجود میان برنامک‌های مورد بررسی، خصوصا در روش‌های مبتنی بر تحلیل ایستا، ناشی از کدهای کتابخانه‌ای خواهد بود. از سوی دیگر، تشخیص مبهم‌نگاری در کدهای مورد توسعه توسط متقلب، نیازمند ویژگی‌هایی از برنامک مورد نظر است که مقاومت بالایی در برابر مبهم‌نگاری داشته باشند. بدین معنا که متقلب برای تغییر این دسته ازویژگی‌ها ناچار به پرداخت هزینه‌ی زمانی و فنی باشد و در نهایت از تغییر این دست از ویژگی‌ها، پرهیز کند. در بسیاری از روش‌های ارائه‌شده در سال‌های اخیر، تشخیص برنامک‌های بازبسته‌بندی شده مبتنی بر ویژگی‌هایی صورت گرفته است که در عین مقاومت در مقابل مبهم‌نگاری، هزینه‌ی محاسباتی تشخیص برنامک‌های بازبسته‌بندی شده را افزایش می‌دهد به طوری که استفاده از این روش‌ها را عملا در یک محیط صنعتی غیر ممکن ساخته‌است.

با توجه به اهمیت تشخیص مبهم‌نگاری و در نهایت تشخیص برنامک‌های بازبسته‌بندی شده و همچنین،

در نظر گرفتن سرعت تشخیص به عنوان یک عامل مهم، در این فصل به بررسی و مرور کارهایی می‌پردازیم که روش‌های گوناگونی را برای تشخیص برنامک‌های بازبسته‌بندی استفاده کرده‌اند و مزایا و معایب هر کدام را به صورت جدا بررسی خواهیم‌کرد. از آن‌جایی که هدف این پژوهش بهبود کارایی روش‌های تشخیص برنامک‌های بازبسته‌بندی شده‌است و تمرکز پژوهش بر روی تشخیص کدهای کتابخانه‌ای نبوده‌است، در ابتدا روند کلی تشخیص برنامک‌های بازبسته‌بندی شده را در پژوهش‌های مرتبط بیان کرده و به اختصار، روش‌های جداسازی کتابخانه‌های اندرویدی از کدهای مورد توسعه را توضیح می‌دهیم و از مرور کارهای پیشین انجام‌شده در این حوزه عبور خواهیم کرد.

در ادامه ابتدا به روند کلی تشخیص برنامک‌های بازبسته‌بندی شده می‌پردازیم و مسئله‌ی تشخیص برنامک‌های بازبسته‌بندی شده را از دیدگاه این پژوهش، شرح می‌دهیم. همچنین، دسته‌بندی انواع روش‌های تشخیص را با توجه به پژوهش‌های سال‌های اخیر بیان می‌کنیم و از هر دسته، چند پژوهش انجام‌شده را بررسی خواهیم کرد. برای درک بهتر روش پیشنهادی، در هر قسمت به بیان مزایا و معایب هر روش خواهیم پرداخت و علاوه بر این روش تشخیص کدهای کتابخانه‌ای در هر پژوهش را مشخص خواهیم‌کرد.

۳ـ۱ تعریف مسئله

علی‌رغم پژوهش‌های متعدد صورت‌گرفته در این زمینه، همانند تعریف بازبسته‌بندی، هنوز تعریف مشخصی نیز برای تشخیص بازبسته‌بندی ارائه‌نشده است. پژوهش‌های سال‌های اخیر در حالت کلی تشخیص بازبسته‌بندی را به دو صورت تعریف می‌کنند:

تعریف ۳ـ۱ (تشخیص بازبسته‌بندی مبتنی بر برنامک مبدا) */ تشخیص بسته‌ی بازبسته‌بندی شده، یعنی تشخیص جفتی از برنامک‌های درون مخزن که دقیقا جفت مشابه برنامک ورودی باشد. به بیان دیگر در این تعریف مشخص می‌شود که برنامک ورودی بازبسته‌بندی شده است یا خیر و در صورتی که بود، جفت برنامک آن درون مخزن نیز مشخص می‌شود.*

تعریف ۳ـ۲ (تشخیص بازبسته‌بندی مبتنی بر تصمیم‌گیری برنامک مقصد) *تشخیص بسته‌ی بازبسته‌بندی شده، یعنی مشخص کنیم برنامک ورودی بازبسته‌بندی شده است یا خیر. در این حالت تشخیص برنامک اصلی اهمیتی ندارد و مسئله، تصمیم‌گیری [۱] درباره‌ی بازبسته‌بندی بودن یک برنامک ورودی است.*

در سال‌های اخیر، اکثر پژوهش‌ها از یکی از تعاریف بالا برای تشخیص بازبسته‌بندی استفاده کرده‌اند. برای پاسخ به تعریف ۲، پژوهش‌هایی نظیر [۳۲، ۳۳، ۳۴] از روش‌های مبتنی بر مدل‌های یادگیری ماشین[۲]

[۱] Decision
[۲] Machine Learning

برای تشخیص برنامک‌های بازبسته‌بندی شده استفاده کرده‌اند. حال آن‌که پژوهش‌های مرتبط با تعریف ۱، نظیر [۳۵، ۳۶] بیشتر از روش‌های مقایسه‌ی دودویی و مبتنی بر شباهت‌سنجی استفاده کرده‌اند.

تعریفی که در این پژوهش مورد استفاده قرارگرفته است، تعریف ۱ است. یعنی تشخیص بازبسته‌بندی منوط به **تشخیص جفت برنامک اصلی** در مخزن برنامک‌های موجود می‌باشد. بنابراین در طی فرایند تشخیص به دو سوال اساسی پاسخ می‌دهیم:

- آیا برنامک ورودی بازبسته‌بندی شده‌ی یک برنامه‌ی دیگر است؟
- در صورتی که برنامک مورد بررسی، بازبسته‌بندی شده‌ی برنامک دیگری بود، آن‌گاه جفت بازبسته‌بندی شده‌ی برنامک ورودی کدام برنامک است.

WW

۳ـ۲ روند کلی تشخیص برنامک‌های بازبسته‌بندی شده

با بررسی پژوهش‌های صورت‌گرفته در حوزه‌ی تشخیص برنامک‌های بازبسته‌بندی شده، درمی‌یابیم که به طور مشخص عمده‌ی این روش‌ها مراحل مشابهی را برای حل این مسئله، دنبال کرده‌اند. به طور کلی عمده‌ی روش‌های تشخیص، به عنوان ورودی، یک برنامک اندویدی شامل یک فایل با پسوند apk را دریافت کرده و پس از گذر از سه مرحله، مسئله را حل می‌کنند. در ادامه به بررسی این سه مرحله می‌پردازیم.

۳ـ۲ـ۱ پیش‌پردازش برنامک‌های اندرویدی

یکی از مراحل مهم در تشخیص برنامک‌های بازبسته‌بندی شده، مرحله‌ی پیش‌پردازش[۳] است که تاثیر به سزایی در سرعت و دقت روش تشخیص خواهد داشت. حذف کدهای کتابخانه‌ای ، حذف کدهای مرده و یا بیهوده و اعمال فیلترهای ساختاری[۴] از موارد نمونه در قسمت پیش‌پردازش است. در این قسمت روش‌های کلی مورد استفاده توسط پژوهش‌های اخیر جهت حذف کدهای کتابخانه‌ای را توضیح می‌دهیم. با توجه به مرور کارهای پیشین انجام‌شده در این حوزه، به صورت کلی دو دیدگاه در مورد تشخیص و جداسازی کتابخانه‌های اندرویدی وجود دارد:

---

[۳] Pre process
[۴] Structural

- **مبتنی بر لیست سفید:** در این روش، لیستی از نام بسته‌های مشهور کتابخانه‌ای در برنامک‌های اندرویدی در دسترس است و با استفاده از نام بسته‌های موجود در برنامک، کدهای کتابخانه‌ای از کدهای مورد توسعه جدا می‌شوند. راه حل‌های مبتنی بر این روش، عموماً در مقابل مبهم‌نگاری‌های ساده‌ای نظیر تغییر نام بسته نیز مقاوم نیستند و به راحتی می‌توان آن‌ها را دور زد. مزیت این روش آن‌است که سرعت بالایی دارد چرا که فقط نام بسته‌ها با یکدیگر مقایسه می‌شوند اما دقت خوبی را ارائه نمی‌دهند.غالب پژوهش‌های مبتنی بر استفاده از لیست سفید، فرض کرده‌اند که تنها کدهای مورد توسعه توسط متقلب مبهم‌نگاری شده‌است و ابهام در کدهای کتابخانه‌ای را نادیده گرفته‌اند.
- **مبتنی بر شباهت‌سنجی و کدهای تکراری:** در این روش، ابتدا مخزن بزرگی از کتابخانه‌های اندرویدی تهیه می‌شود و به روش‌های گوناگون کدهای کلاسی برنامک و کدهای کتابخانه‌ای موجود در مخزن، با یکدیگر مقایسه می‌شوند و بدین طریق کتابخانه‌های اندرویدی از کدهای مورد توسعه در برنامک، جدا می‌شود. روش‌های مبتنی بر شباهت‌سنجی، بسته به این‌که از چه روشی برای یافتن کدهای تکراری استفاده می‌کنند، دقت‌های متفاوتی دارند اما به صورت کلی می‌توان گفت که مقاومت آن‌ها در مقابل مبهم‌نگاری بسیار بیشتر از روش‌های مبتنی بر لیست سفید است چرا که در صورتی که ویژگی‌های منتخب مقابل مبهم‌نگاری مقاوم باشند، آن‌گاه می‌توان گفت که درصد بالایی از کتابخانه‌های اندرویدی را می‌توان از کد اصلی برنامک جدا کرد.

۳ـ۲ـ۲ استخراج ویژگی

پس از حذف کدهای کتاب‌خانه‌ای در قسمت قبلی و انجام پیش‌پردازش‌های مورد نیاز، کدهای منبع برنامک هدف، به یک طرح کلی مدل می‌شود. به صورت کلی می‌توان روش‌های تشخیص برنامک‌های بازبسته‌بندی شده را در پژوهش‌های سالیان اخیر، ناشی از تفاوت در دیدگاه در مرحله‌ی استخراج ویژگی[۵] دانست. همانطور که در شکل ۳ـ۱ مشاهده می‌شود، روش‌های تشخیص برنامک‌های بازبسته‌بندی به صورت کلی به دو بخش تحلیل ایستا و تحلیل پویا تقسیم می‌شود. از آن‌جایی که هدف ما در این پژوهش، تنها بررسی پژوهش‌هایی است که روش‌های تشخیص بازبسته‌بندی ارائه داده‌اند بنابراین روش‌هایی که توسعه‌دهندگان و شرکت‌های توسعه‌دهنده جهت جلوگیری از انجام بازبسته‌بندی پیاده‌سازی می‌کنند را به صورت خلاصه‌تر شرح خواهیم‌داد. به صورت کلی، می‌توان روش‌های تشخیص برنامک‌های بازبسته‌بندی شده را به دو بخش روش‌های تحلیل پویا و یا روش‌های تحلیل ایستا تقسیم کرد که در ادامه به بررسی هر کدام از این روش‌های می‌پردازیم.

[۵] Feature Extracting

- **روش‌های مبتنی بر تحلیل ایستا:** روش‌های مبتنی بر تحلیل ایستا، در مقابل مبهم‌نگاری‌های ایستا که در هنگام بازبسته‌بندی و انجام دی‌کامپایل انجام می‌شود مقاوم هستند. اما همانطور که می‌توان حدس زد، این دسته از روش‌ها مقابل روش‌های مبهم‌نگاری همانند بازتاب مقاومتی ندارند و ممکن است دچار خطا شوند. همچنین روش‌های مبهم‌نگاری مبتنی بر رمزنگاری پویا نیز این روش‌ها را دچار خطا می‌کند. یکی از مزایای مهم روش‌های مبتنی بر تحلیل ایستا آن است که در صورت پیاده‌سازی درست و استفاده از ویژگی‌های مقاوم، می‌توانند طیف وسیعی از برنامک‌های بازبسته‌بندی شده را تشخیص دهند.

- **روش‌های مبتنی بر تحلیل پویا:** ارائه‌ی روش‌های مبتنی بر تحلیل پویا، به هدف جلوگیری از مبهم‌نگاری‌های در لحظه‌ی اجرا[۶] که در برنامک‌های اندرویدی صورت می‌گیرد، می‌باشد. به همین علت روش‌های موجود در این حوزه، عمدتا برنامک‌ها را در هنگام اجرا بررسی و استخراج ویژگی عمدتا در هنگام اجرا انجام می‌گیرد. به طول کلی، روش‌های مبتنی بر تحلیل پویا از مقاومت بیشتر در مقابل استفاده از راهکارهای مبهم‌نگاری برخوردار هستند. استفاده از شبیه‌سازهای جعبه‌شن[۷] به وفور در پژوهش‌های این حوزه، یافت می‌شود. یکی از چالش‌های اصلی در تشخیص برنامک‌های اندرویدی بازبسته‌بندی شده، چگونگی پیاده‌سازی شبیه‌ساز ها[۸]ست. بسیار از شبیه‌سازها توانایی شبیه‌سازی تمامی خدمات موجود در برنامک را ندارند و برای تحلیل دقیق‌تر نیازمند استفاده از کاربران واقعی در شبیه‌سازی و استفاده از خدمات برنامک هستند. عامل دیگری که تشخیص با استفاده از تحلیل پویا را مشکل می‌کند، این است که بسیاری از بدافزارهای توسعه‌یافته، توانایی تشخیص محیط اجرای شبیه‌سازی‌شده را دارند و ممکن است تمامی قابلیت‌های خود و یا بخشی از آن را به جهت دور زدن سیستم‌های تشخیص پویا، پنهان کنند.

۳ـ۲ـ۳ تشخیص بازبسته‌بندی

در این مرحله با توجه به معیارها و ویژگی‌هایی که از قسمت قبل به دست آمده است و با استفاده از روش‌های گوناگون برنامک بازبسته‌بندی شده مشخص می‌شود. به صورت کلی، روش‌های پیاده‌سازی شده در این قسمت، مبتنی بر مقایسه‌ی دودویی و یا طبقه‌بندی و یادگیری ماشین هستند.

- **مقایسه‌ی دودویی:** روش‌های مبتنی بر مقایسه‌ی دودویی، مدل استخراج شده در قسمت قبلی را با استفاده از شباهت‌سنجی با برنامک‌های موجود در مخزن مقایسه می‌کند و در نهایت برنامک

---

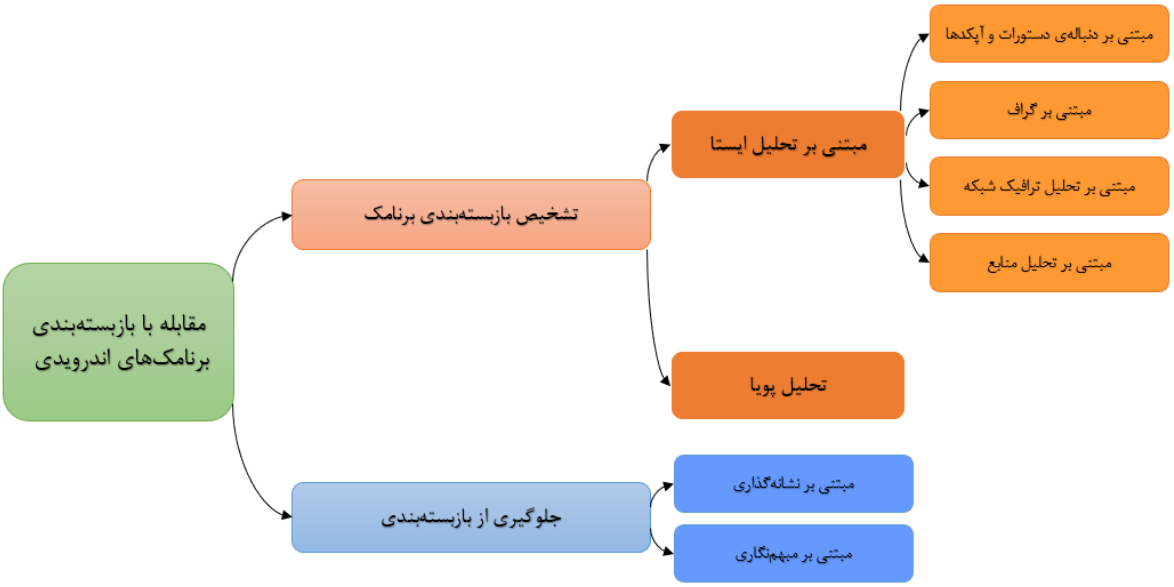
[۶] Execution Time
[۷] Sand Box
[۸] Simulator

بازبسته‌بندی شده را مشخص می‌کند. اکثر روش‌های مبتنی بر مقایسه‌ی دودویی، جفت برنامک اصلی را نیز مشخص می‌کنند و از تعریف ۳ـ۱ استفاده می‌کنند بنابراین یکی از مزیت‌های این روش‌ها پوشش گسترده‌تر از تعریف تشخیص بازبسته‌بندی است ولی در کنار آن اکثر روش‌های موجود در این زمینه، محاسبات بالایی دارند که باعث می‌شود سرعت آن‌ها کاهش یابد.

- **مبتنی بر طبقه‌بندی و یادگیری ماشین:** یکی دیگر از روش‌های تشخیص بازبسته‌بندی با استفاده از ویژگی‌های مستخرج از مرحله‌ی قبل، استفاده از طبقه‌بند ها و مدل‌های یادگیری ماشین است. اکثر پژوهش‌های موجود در این زمینه از تعریف ۳ـ۲ برای تشخیص برنامک بازبسته‌بندی شده استفاده می‌کنند. بنابراین، تنها تصمیم‌گیری در مورد بازبسته‌بندی بودن یا نبودن برنامک ورودی را انجام می‌دهند. یکی از مزایای مهم این روش‌ها، سرعت بالای آن است چرا که تنها در زمان مرحله‌ی یادگیری، نیازمند محاسبات بالایی هستند و در صورتی که مدل این روش‌ها به درستی عمل کند، سرعت تشخیص به صورت قابل توجهی بالاتر از روش‌های مبتنی بر مقایسه‌ی دودویی است.

۳ـ۳ روش‌های تشخیص بازبسته‌بندی

همانطور که در شکل ۳ـ۱ مشاهده می‌شود، اکثر پژوهش‌های تشخیص بازبسته‌بندی از روش‌های مقایسه‌ای مبتنی بر تحلیل ایستا و پویا استفاده می‌کنند. در ادامه‌ی این قسمت ابتدا روش‌های ایستا و همچنین پژوهش‌های اخیر مرتبط با این حوزه را بررسی خواهیم کرد و در ادامه روش‌های مبتنی بر تحلیل پویا شرح داده می‌شود.



شکل ۳ـ۱: شمای کلی امضای متد در پژوهش

۳ـ۳ـ۱ مبتنی بر تحلیل ایستا

همانطور که گفتیم تحلیل ایستا، روشی محبوب در میان پژوهش‌های اخیر موجود در این حوزه است چرا که پیچیدگی‌های روش‌های پویا را ندارد و می‌توان به کمک آن‌ها طیف وسیعی از تشخیص مبهم‌نگاری‌ها را در برنامک‌های اندرویدی بازبسته‌بندی شده پشتیبانی کرد.

روش‌های مبتنی بر آپکد و دستورات

استفاده از آپکد[۹]های موجود در فایل‌های دالویک، یکی از روش‌های تشخیص برنامک‌های بازبسته‌بندی شده است. هدف از پژوهش آقای ژو و همکاران [۳۷]، توسعه‌ی ابزاری به نام درویدمس[۱۰] بوده است که توسط آن مشخص شود چه تعدادی از برنامک‌های موجود در فروشگاه‌های اندرویدی غیررسمی، بازبسته‌بندی شده‌ی برنامک‌های موجود در فروشگاه‌های رسمی هستند. همانطور که گفته شد نظارت کافی‌ای بر روی فروشگاه‌های غیر رسمی وجود ندارد، بنابراین متقلبین از این فروشگاه‌ها به عنوان یک راه امن و دردسترس برای پخش‌کردن برنامک‌های بازبسته‌بندی شده استفاده می‌کنند. برای استخراج امضای برنامک در این پژوهش از کدهای دالویک موجود در `Classes.dex` و امضای دیجیتال برنامه‌نویس در

[۹] Opcode
[۱۰] DroidMoss

فراداده[11] استفاده‌شده است. پس از جداسازی کدهای کتابخانه‌ای به وسیله‌ی لیست سفید و استخراج آپکدها از فایل‌های دالویک، از یک پنجره‌ی لغزان[12] روی آپکدها استفاده شده و در نهایت چکیده[13]ی آپکدها به همراه امضای دیجیتال برنامه‌نویس، موجود در پوشه‌ی META-INF تشکیل امضای برنامک را می‌دهند. همانطور که می‌توان فهمید، فرض پژوهش این بوده است که کلید خصوصی توسعه‌دهنده لو نرفته‌است. در نهایت برای قسمت شباهت‌سنجی، از الگوریتم فاصله ویرایشی[14] استفاده‌شده است. در قسمت شباهت‌سنجی از ۲۲۹۰۶ برنامک موجود در فروشگاه‌های رسمی استفاده شده و نتایج پژوهش نشان می‌دهد که ۵ تا ۱۳ درصد از برنامک‌های موجود در فروشگاه‌های غیر رسمی، بازبسته‌بندی شده‌ی برنامک‌های فروشگاه‌های رسمی است. در پژوهش دیگری که توسط آقای ژو[۲۸] ارائه شده‌است، هدف پژوهش، افزایش سرعت پژوهش قبلی با استفاده از نمونه‌های n تایی از آپکدها بوده است. در این پژوهش امضای هر برنامک متشکل از قسمتی از فراداده‌ی آن شامل فایل‌های منیفست[15] و اطلاعاتی در مورد تعداد فایل‌های برنامک، توصیفات آن و چکیده‌ی آپکدهای دستورات برنامه است. این پژوهش با استفاده از یک مرحله پیش‌پردازش شامل بررسی فایل فراداده‌ی برنامک‌های موجود،فضای جست‌وجوی دودویی برنامک‌های مورد مقایسه را کاهش می‌دهد.دزنوز و همکاران[۳۸] روش‌دیگری را مبتنی بر شباهت‌سنجی روی آپکدها با استفاده از فاصله‌ی فشرده‌سازی نرمال‌شده ارائه کرده‌اند. در این پژوهش ابتدا برای هر متد با توجه به دنباله‌ی دستورات موجود امضای مشخصی تولید می‌شود و در مرحله‌ی بعد متدهایی که بکتا هستند از هر دو برنامک، بر اساس معیار فاصله‌ی فشرده‌سازی نرمال‌شده با یکدیگر مقایسه و بدین ترتیب متدهای مشابه استخراج می‌شود. در پژوهش [۳۹، ۴۰] پس از استخراج هیستوگرام‌های مربوط به تکرار آپکدها در قسمت‌های مختلف برنامک، هیستوگرام‌ها با استفاده از معیار فاصله‌ی مینی‌کاوسکی که یک معیارفاصله‌ی مبتنی بر هیستوگرام‌ها است مقایسه می‌شوند و در نهایت برنامک‌های بازبسته‌بندی شده مشخص می‌شوند. جرومه و همکاران[۴۱] در پژوهش خود با استفاده از آپکدها و تکرار آن‌ها و روش‌های مبتنی بر یادگیری ماشین برنامک‌های بازبسته‌بندی شده را تشخیص می‌دهند.در پژوهشی که توسط سرنیل و همکاران [۴۲]، ارائه‌شده است، از نمونه‌برداری مبتنی بر $n-$ گرام در ۴ اندازه‌ی متفاوت ۱ تا ۴ استفاده شده‌است. برای شباهت‌سنجی از روش‌های طبقه‌بندی مبتنی بر درخت‌تصمیم، شبکه‌های عصبی و بردار ماشین استفاده شده‌است.

آقای لین و همکاران[۴۳] در پژوهش خود، با استفاده از فراخوانی‌های سیستمی[16] صدا زده‌شده توسط برنامک، رفتار آن را طبقه‌بندی می‌کنند. به عقیده‌ی این پژوهش، از آن‌جایی که اکثر بدافزارهای هم‌خانواده،

[11] MetaData
[12] Sliding Window
[13] Hash
[14] Edit Distance
[15] Manifest
[16] System calls

در بازبسته‌بندی برنامک‌های اندرویدی، رفتار مشابه یکدیگر دارند، بنابراین استفاده از فراخوانی‌های سیستم و استخراج آن‌ها از سطح بایت‌کدهای دالویک و سطح نخ[۱۷]، می‌تواند امضاء یکتایی از هر برنامک تولید کند. پس از استخراج بردار ویژگی[۱۸] از فراخوانی‌های موجود با استفاده از آپکدهای برنامک، از یک طبقه‌بند بیز برای شباهت‌سنجی استفاده شده‌است. با توجه به روش پژوهش، شناسایی و طبقه‌بندی بدافزارهای بازبسته‌بندی شده‌ای که رفتار مشخصی ندارند و در مخزن بدافزارها موجود نیستند، یکی از ویژگی‌های مفید پژوهش ارائه‌شده است. فروکی و همکاران [۴۴] در پژوهش خود یک راه حل مبتنی بر استفاده از بلاک‌های ۶۴ بایتی بر روی فایل‌های دودویی برنامک‌های اندرویدی ارائه‌کرده اند. در روش ارائه‌شده پس از استخراج بلاک‌های ۶۴ بایتی از فایل‌ها و با استفاده از چکیده خلاصه تشابه و استخراج آنتروپی[۱۹] برای هر بلاک، بلاک‌هایی که کوچکتر و بزرگتر از یک حد کمینه و آستانه باشند حذف می‌شوند. سپس به هر بلاک با توجه آنتروپی آن، یک اولویت[۲۰] اختصاص پیدا کرده که نشان می‌دهد بلوک مورد نظر دارای آنتروپی با احتمال بیشتر است. در نهایت پس از حذف بلوک‌هایی که احتمال رخداد پایین‌تری دارند نرخ مثبت غلط پژوهش کاهش یافته و از یک روش مبتنی بر بلوم‌فیلتر[۲۱] برای مقایسه و شباهت‌سنجی استفاده می‌شود.

آقای کو و همکاران[۴۵] از یک راه‌حل مبتنی بر استفاده از $k-$گرام برای تشخیص بسته‌های بازبسته‌بندی شده استفاده کرده‌اند. نویسندگان، از حذف عملوندهای[۲۲] موجود در کدهای دودویی، به جهت کاهش مثبت‌های غلط در تشخیص بسته‌های بازبسته‌بندی شده استفاده کرده‌اند. در پژوهش کیشو و همکاران [۴۶] از یک راه‌حل مبتنی بر ترکیبی از دستورات کلاسی و متدهای برنامک استفاده کرده‌اند. در این پژوهش، در دو مرحله، ابتدا کلاس‌های مشابه با یکدیگر مشخص می‌شود و سپس در داخل کلاس‌های مشابه، متدهایی که یکسان هستند یافت می‌شود. شباهت‌سنجی میان کلاس‌ها، با استفاده از سه ویژگی، شامل لیست تمامی متدهای کلاس شامل ورودی و خروجی، لیست متغیرهای کلاسی و لیست کلاس‌هایی که داخل این کلاس فراخوانی شده‌اند، انجام می‌شود. پس از استخراج کلاس‌های مشابه، برای یافتن متدهای مشابه میان دو کلاس، از یک امضای مشترک شامل توصیف متدها به همراه نوع ورودی و خروجی آن‌ها و همچنین نام آن‌ها استفاده می‌شود. شباهت‌سنجی با استفاده از فاصله‌ی فشرده‌سازی[۲۳] انجام شده‌است.

راهول و همکاران[۴۷]، روشی را پیشنهاد کرده‌اند که در آن استخراج ویژگی مبتنی بر درخت نحو انتزاع[۲۴] انجام می‌شود. ابزار پیشنهادی در این پژوهش پس از دستیابی به کد میانی برنامک‌های اندرویدی و

---

[۱۷] Thread
[۱۸] Feature Vector
[۱۹] Entropy
[۲۰] Priority
[۲۱] Bloom filter
[۲۲] Operand
[۲۳] Compression Distance
[۲۴] Abstract Syntax Tree

تبدیل آن به مجموعه‌ای از قوانین نحوی، که به صورت مجموعه‌ای از عبارات منظم[۲۵] هستند، درخت نحو انتزاع را در سطح تابع تشکیل می‌دهد و سه ویژگی تعداد ورودی هر تابع، نام توابع صدازده‌شده به صورت مستقیم و مجازی[۲۶] و متغیرهای شرطی را استخراج می‌کند. سپس برای طبقه‌بندی از الگوریتم نزدیک‌ترین همسایه به جهت تشخیص بازبسته‌بندی استفاده شده‌است. نرخ منفی غلط بسیار پایین از ویژگی‌های مورد توجه این پژوهش است. همچنین برای ذخیره‌سازی درخت نحو انتزاع، از یک ساختار مبتنی بر درخت $B+$ و پایگاه‌داده‌ی $MySql$ استفاده شده‌است.

به صورت کلی می‌توان گفت که روش‌های مبتنی بر دستورات، خصوصا روش‌هایی که به صورت مستقیم از آپکد برای تشخیص برنامک‌های بازبسته‌بنده شده استفاده می‌کنند، توانایی بالایی را در تشخیص برنامک‌های بازبسته‌بندی شده ارائه نمی‌دهند. این روش‌ها هم‌اکنون مقابل ساده‌ترین مبهم‌نگاری‌ها نظیر تغییر نام بسته‌ها و کلاس‌ها مقاوم نیستند و بخش زیادی از پژوهش‌های این حوزه، بازبسته‌بندی را بدون تغییر در کدهای برنامک اصلی تعریف کرده‌اند که به نظر با توجه به وجود مبهم‌نگارهای امروزی، این فرضی غلط و غیر قابل اتکا است.

روش‌های مبتنی بر گراف

به صورت کلی می‌توان پژوهش‌های صورت‌گرفته در دسته روش‌های مبتنی بر گراف، را از دو جنبه بررسی کرد. دیدگاه اول پژوهش‌هایی هستند که در نهایت امضاء هر برنامک را با استفاده از یک مدل گرافی نشان می‌دهند. در این دسته از پژوهش‌ها، برای مقایسه‌ی امضا، ناچاراً از الگوریتم‌های تشابه گراف نظیر الگوریتم‌های تشخیص گراف‌های هم‌ریخت استفاده می‌شود و به علت سربار محاسباتی بسیار بالای این پژوهش‌ها، روش‌های این دسته بسیار کند هستند. دیدگاه دوم پژوهش‌هایی هستند که صرفاً با بررسی ویژگی‌های مبتنی بر گراف‌های جریان و داده‌ای میان قسمت‌های مختلف، ویژگی‌های هر برنامک را استخراج کرده و در نهایت امضاء هر برنامک را تشکیل می‌دهند. همانطور که می‌توان حدس زد، دسته‌ی دوم از سرعت بالاتری در تشخیص برخوردار است اما چگونگی مدل‌سازی با استفاده از ویژگی‌های گرافی، بخش مهمی در پژوهش‌های این دسته است که باید به دقت پیاده‌سازی شود.

در پژوهشی که توسط آقای کروسل و همکاران [۴۸]، ابزاری مبتنی بر گراف وابستگی توسعه‌داده شده‌است. در این ابزار ابتدا، برنامک‌های موجود در مخزن با استفاده از یک ابزار شباهت‌سنجی در سطح فراداده[۲۷]ی برنامک، به جهت افزایش سرعت، کاهش‌می‌یابد. پس از حذف کدهای کتابخانه‌ای به روش لیست سفید، امضاء هر برنامک با استفاده از گراف وابستگی استخراج‌شده تشکیل می‌گردد. گراف وابستگی

[۲۵] Regular Expressions
[۲۶] Virtual
[۲۷] Meta Data

توابع، وابستگی اجزای یک تابع از دو منظر کنترلی و داده‌ای را معرفی می‌کند. وابستگی کنترلی[۲۸] در این پژوهش، الزام اجرای یک دستور خاص پیش از دستور دیگری است و وابستگی داده‌ای[۲۹]، الزام مقداردهی متغیر پیش از اجرای دستور مرتبط با آن است. در قسمت شباهت‌سنجی پس از ساخت گراف وابستگی داده‌ای، با استفاده از الگوریتم تشخیص گراف‌های هم‌ریخت VF2[۳۰] شباهت‌سنجی انجام شده و بسته‌های بازبسته‌بندی شده مشخص می‌شوند. در پژوهش دیگری که توسط آقای سان[۱۰] انجام شده‌است، هدف پژوهش افزایش دقت تشخیص برنامک‌های بازبسته‌بندی شده با تاکید بر شبیه‌سازی رفتار برنامک بوده‌است. در این پژوهش، واسط‌های برنامه‌نویسی برنامک‌های اندرویدی مشخص کننده‌ی رفتار اصلی برنامک در نظر گرفته‌شده است. برای ساخت گراف هر برنامک، از گراف جریان مبتنی بر فراخوانی واسط‌های اندرویدی استفاده شده و در نهایت پس از استخراج گراف، هر گراف نمایانگر امضاء یک برنامک می‌باشد. در گراف حاصل هر گره گراف حاوی اطلاعات یک واسط و یال‌های گراف شامل جریان کنترلی بین واسط‌های اندرویدی است. برای شباهت‌سنجی، از الگوریتم $VF2$ جهت تشخیص گراف‌های هم‌ریخت استفاده شده‌است. در مرحله‌ی آخر برنامک‌هایی که امضاء مشابه و یکسانی در تشخیص هم‌ریختی دریافت کرده‌اند به عنوان برنامک‌های بازبسته‌بنده شده در نظر گرفته می‌شوند.

پژوهش آقای هو و همکاران[۴۹] شامل دو مرحله‌ی ساخت گراف فراخوانی متدهای[۳۱] برنامک و ماژول تشخیص بازبسته‌بندی است. پس از دیس‌اسمیل کردن فایل‌های برنامک، گراف فراخوانی متدهای برنامک تشکیل شده و تشکیل جنگلی از گراف‌های متصل و جدا از هم می‌دهند.سپس با استفاده از فراخوانی واسط‌های اندرویدی موجود در هر متد، گراف به دو بخش فراخوانی‌های حساس[۳۲] و غیرحساس[۳۳] تقسیم می‌شود و با توجه به میزان حساسیت واسط‌های فراخوانی‌شده، امتیاز اولویت[۳۴] به هر گراف نگاشت می‌شود و در نهایت با استفاده از مقایسه‌ی گرافی مبتنی بر امتیاز اولویت، شباهت‌سنجی انجام می‌شود.

از آن‌جایی که پژوهش‌های مبتنی بر گراف در تشخیص برنامک‌های بازبسته‌بندی، عمدتا به دلیل استفاده از روش‌های تشخیص گراف‌های همریخت کند هستند، ژو و همکاران [۵۰] روشی برای افزایش سرعت در تشخیص ارائه کرده‌اند. در این پژوهش در ابتدا ماژول‌های اصلی برنامک که رفتار اصلی آن را شکل می‌دهند شناسایی می‌شود. برای شناسایی ماژول‌های اصلی برنامک، از یک گراف جهت‌دار مبتنی بر ارتباط بسته[۳۵]های برنامک با یکدیگر استفاده شده و در نهایت یال‌های گراف بر اساس میزان ارتباطات بین بسته‌ها، مقداردهی می‌شود. با تشکیل گراف وزن‌دار ابتدایی، بسته‌هایی که ارتباط آن‌ها بر اساس وزن یال بین دوبسته

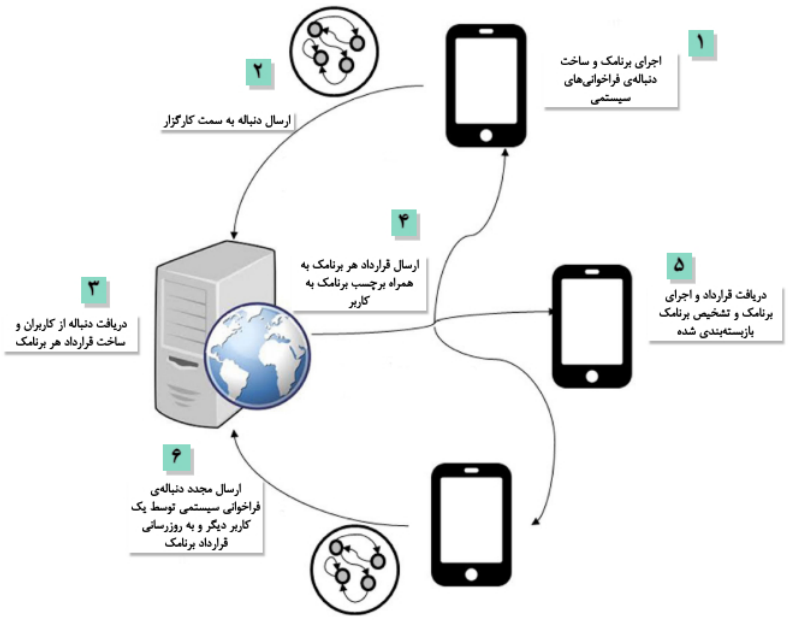
[۲۸] Control Dependency
[۲۹] Data Dependency
[۳۰] Graph Isomorphism
[۳۱] Method Invocation Graph
[۳۲] Sensitive
[۳۳] Non-Sensitive
[۳۴] Priority Score
[۳۵] Package

از یک مقدار آستانه بیشتر باشد، با یکدیگر ادغام می‌شوند و رویه‌ی ادغام بسته‌ها در یک روند بازگشتی تا زمانی که هیچ بسته‌ای را نتوان با یکدیگر ادغام کرد تکرار می‌شود. در این حالت بسته‌ی نهایی شامل بسته‌ی اصلی برنامک است که منطق برنامک در این قسمت پیاده‌سازی شده‌است. رویه‌ی ساخت گراف ارتباطی بین بسته‌ها و ایده‌ی استفاده شده در قسمت ادغام بسته‌های اصلی با یکدیگر، ایده‌ای نو در این حوزه است که منجر به افزایش سرعت تشخیص نسبت به تمامی روش‌های گرافی شده‌است. برای مقایسه‌ی میان ماژول‌های اصلی برنامک، ابتدا اصلی‌ترین ماژول شامل بیشترین تعداد فعالیت، مشخص می‌شود و مقایسه میان ماژول‌های اصلی برنامک‌های اندرویدی، با استفاده از یک بردار ویژگی متشکل از فراخوانی واسط‌ها و مجوزهای[۳۶] درخواستی انجام می‌شود. برای مقایسه از درخت VP[۳۷] استفاده شده‌است که منجر به افزایش سرعت در کنار دقت مناسب شده‌است. برخلاف روش‌های معمول گرافی و در نهایت مقایسه‌ی دودویی، روش پیاده‌سازی شده در پژوهش ژو، با مرتبه‌ی زمانی $n.logn$ یکی از پر سرعت‌ترین روش‌های مبتنی بر تشکیل گراف می‌باشد.

در پژوهش دیگری که به جهت افزایش سرعت در دسته پژوهش‌های مبتنی بر گراف ارائه شده‌است، چن و همکاران[۵۱] با استفاده از مدل کردن گراف به یک فضای سه‌بعدی، سرعت تشخیص و مقایسه‌ی گراف‌های هم‌ریخت را به مراتب افزایش داده‌اند. ایجاد کدهای مرده[۳۸] در کدهای برنامک، منجر به تغییر گراف جریان برنامک‌های اندرویدی می‌شود به همین جهت در این پژوهش تمامی گره‌های گرافی که نشان‌دهنده‌ی متدهای برنامک هستند به یک فضای سه‌بعدی نگاشت شده و مرکز جرم هر گراف با توجه به مختصات گره‌های گرافی تعیین می‌شود. در قسمت مقایسه‌ی گرافی، مرکز جرم گراف‌های متناظر با یکدیگر مقایسه شده و کاندیدهای بازبسته‌بندی مشخص می‌شود. در مرحله‌ی بعد برای مقایسه‌ی گره‌های هر گراف و تطبیق گراف‌های کاملا متناظر، از مقایسه‌ی فاصله‌ی ویرایشی گره‌های متناظر استفاده می‌شود. روش ارائه‌شده علاوه بر مقاومت بالا مقابل مبهم‌نگاری، ناشی از مدل‌کردن برنامک در یک فضای گرافی، به دلیل استفاده از روشی نو در مقایسه‌ی گراف‌های مخزن برنامک‌ها، سرعت بسیار بالاتری از روش‌های پیشین دارد. برای حذف کتاب‌خانه‌های اندرویدی از روش لیست سفید مبتنی بر اندازه‌ی بسته‌های مورد مقایسه استفاده شده‌است.

بر خلاف پژوهش‌های رایج در حوزه‌ی تشخیص بازبسته‌بندی برنامک‌های اندرویدی، در پژوهش آقای آلدینی و همکاران[۳]، از یک معماری کارخواه‌ـ‌کارگزار[۳۹] مطابق با شکل ۳ـ۲، برای تشخیص برنامک‌های اندرویدی بازبسته‌بندی شده استفاده شده‌است. در این معماری یک برنامک بر روی دستگاه اندرویدی کاربران نصب می‌شود و شروع به ثبت و ارسال فراخوانی‌های سیستمی به کارگزار می‌کند.

[۳۶] Premessions
[۳۷] Vantage-Point Tree
[۳۸] Dead Code
[۳۹] Client-Server



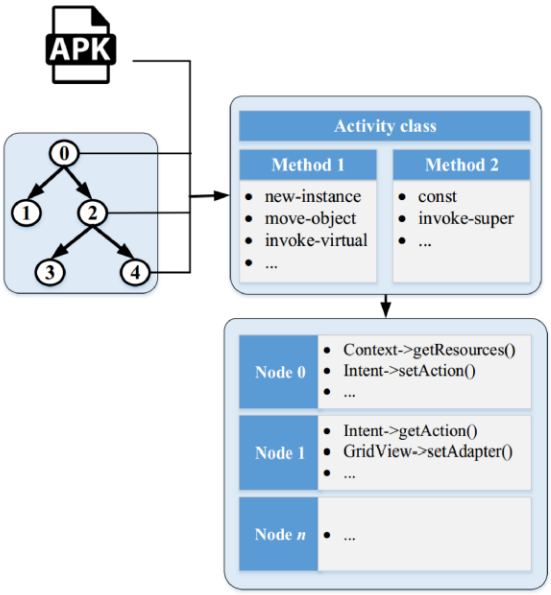
شکل ۳ـ۲: مراحل تشخیص برنامک‌های بازبسته‌بندی شده در پژوهش آقای آلدینی[۳]

ایده‌ی پژوهش، استفاده از فراخوانی‌های سیستمی برای شبیه‌سازی ایستای رفتار برنامک‌های اندرویدی بوده است. اثرانگشت برنامک توسط گراف فراخوانی‌های سیستمی ارسالی از سمت کاربران در سمت کارگزار، تشکیل می‌شود سپس با استفاده از اثرانگشت موجود، یک مدل برنامک تحت عنوان قرارداد ساخته شده و این مدل به برنامه‌ی کارخواه فرستاده می‌شود. در سمت کارخواه، برنامک نصب‌شده در مرحله‌ی اول، فراخوانی‌های سیستمی برنامک موجود را با قرارداد فرستاده‌شده مطابقت می‌دهد و در صورتی که فاکتورهایی نظیر نوع و تعداد فراخوانی‌های اندرویدی برنامک، با مدل پیش‌بینی شده یکسان نباشد، هشدار لازم از طریق برنامک نصب‌شده روی کارخواه به کاربر داده می‌شود. در کنار استفاده از گراف فراخوانی‌های سیستمی، پیاده‌سازی یک معماری کارخواه و کارگزار یکی از ایده‌های نو در این پژوهش بوده‌است. همچنین به دلیل استفاده از این معماری، پردازش سمت کارخواه به حداقل خود رسیده است و تحلیل برنامک نیازمند هیچ دانش اولیه‌ای از سمت کارخواه نمی‌باشد. علاوه بر این، تعداد برنامک‌های موجود در مخزن کارگزار، به صورت مرتب افزایش پیدا کرده و این موجب پویایی مخزن برنامک‌های اندرویدی پژوهش می‌شود.

پژوهش دیگری در زمینه‌ی روش‌های تشخیص بازبسته‌بندی مبتنی بر گراف توسط جنگ و همکاران[۵۲] ارائه شده‌است. در این پژوهش، پس از استخراج گراف ارتباط بین کلاس‌ها از داخل بایت‌کدهای دالویک برنامک اندرویدی، امضای هر برنامک شامل یک بردار ویژگی، متشکل از فراخوانی‌های کلاسی برنامک می‌باشد. مقایسه‌ی بردارهای برنامک‌های موجود به وسیله‌ی الگوریتم یافتن بزرگ‌ترین زیردنباله‌ی مشترک[۴۰]

[۴۰] Longest common Subsequence

انجام شده و تعیین یک حد آستانه در زیردنباله‌های مشترک، برنامک بازبسته‌بندی شده شناسایی می‌شود. روش مورد نظر را می‌توان به نوعی یک روش مبتنی بر گراف و دنباله‌ی آپکدهای برنامک توصیف کرد چرا که از هر دو ایده‌ی مدنظر استفاده نموده‌است. یکی دیگر از آخرین پژوهش‌های موجود در این دسته،در سال ۲۰۲۱ توسط نگویان[۴] مطرح شده‌است. پژوهش مورد نظر مبتنی بر استخراج گراف فعالیت[۴۱] در برنامک‌های اندرویدی است. همانطور که در شکل ۳ـ۳ مشاهده می‌شود، گره‌های گراف مورد نظر شامل لیستی از واسطهای فراخوانی‌شده در آن فعالیت و یال‌های گراف، نشان‌دهنده‌ی یک انتقال از یک فعالیت به فعالیتی دیگر است. یکی از معایب این پژوهش استفاده از الگوریتم VF2 به عنوان الگوریتم اصلی برای مقایسه و یافتن گراف‌های هم‌ریخت است که باعث کاهش سرعت پژوهش شده‌است.



شکل ۳ـ۳: گراف فعالیت و محتوای گره‌های آن در پژوهش نگویان[۴]

به عنوان آخرین پژوهش صورت گرفته در این قسمت، پژوهش[۵، ۵۳] را بررسی خواهیم کرد. پژوهش [۵۳] که الهام‌دهنده‌ی پژوهش [۵] می‌باشد با استفاده از پیمایش گرافی برروی گراف جریان برنامک‌های اندرویدی و استخراج ویژگی‌های متعدد نظیر واسطهای اندرویدی، فراخوانی توابع و استفاده از فیلترهای ساختاری نظیر اندازه‌ی طول امضا، امضای هر کلاس را تشکیل می‌دهد و در نهایت با استفاده از الحاق تمامی کلاس‌های برنامک (مرتب شده به صورت الفبایی) امضاء مخصوص هر برنامک ساخته می‌شود. تفاوت این دوپژوهش بیشتر در ساختار امضاء هر برنامک می‌باشد که در فصل ۴ به توصیف بیشتر این دو پژوهش و شرح تفاوت‌های آن خواهیم پرداخت.

به طور کلی می‌توان گفت که روش‌های گرافی تشخیص برنامک‌های بازبسته‌بندی شده از دقت بالایی در

[۴۱] Activity Graph

تشخیص برخوردار هستند. از طرفی اکثر روش‌های ارائه‌شده در این دسته، خصوصاً آن‌هایی که در نهایت هر برنامک را به یک طرح گرافی مدل می‌کنند، روش‌های تشخیص گراف‌های هم‌ریخت را در قسمت مقایسه به کار گرفته‌اند که این موضوع باعث می‌شود تا سربار محاسباتی سنگینی به پژوهش‌های مطرح وارد و سرعت تشخیص را کند سازد. علاوه بر این همانطور که بررسی شد روش‌های ارائه‌شده، در صورتی که مدل‌های گرافی را در فضای دیگری بررسی کنند، سرعت تشخیص روش بالاتر رفته و می‌توان از دقت در تشخیص مسائل گرافی نیز استفاده نمود.

روش‌های مبتنی بر تحلیل ترافیک شبکه

از آن‌جایی که ترافیک شبکه‌ی[۴۲] عبوری از برنامک‌هایی که مبتنی بر فضای مجازی عمل می‌کنند، می‌تواند رفتار خوبی از برنامک را مدل کند بنابراین استفاده از ویژگی‌های مبتنی بر این ترافیک شبکه، از محبوبیت بالایی میان توسعه‌دهندگان، برخوردار است. به عنوان اولین پژوهش مورد بررسی، پژوهش ارائه‌شده توسط وو و همکاران [۵۴] را بررسی خواهیم کرد. در این پژوهش امضای هر برنامک مبتنی بر ترافیک http تولیدشده توسط آن درست می‌شود. در مرحله‌ی اول این پژوهش، تمامی ترافیک تولیدشده توسط برنامک جمع‌آوری شده و در مرحله‌ی بعدی ترافیک http جداسازی و تحلیل روی این دسته ادامه پیدا می‌کند. ترافیک حاصل از برنامک‌های اندرویدی در این پژوهش به دو دسته‌ی کلی تقسیم می‌شود:

- ترافیک مرجع[۴۳]: ترافیک تولیدشده توسط برنامک توسعه‌یافته
- ترافیک کتابخانه‌ای: ترافیک تولیدشده توسط کدهای کتابخانه‌ای

جهت جداسازی ترافیک مرجع و ترافیک کدهای کتاب‌خانه‌ای از الگوریتم‌های تطبیق جریان ترافیک[۴۴] http و الگوریتم تطبیق هانگرین[۴۵] استفاده شده‌است. در قسمت شباهت‌سنجی از درخت جست‌وجوی VPT به جهت افزایش سرعت به دلیل متوازن بودن درخت، استفاده شده است. پژوهش مورد نظر ایده‌ای نو در زمینه‌ی تشخیص برنامک‌های بازبسته‌بندی شده است اما مشکل اصلی این پژوهش آن است که در مقابل ترافیک رمزنشده مقاوم نیست و عملا در برنامک‌هایی که تمامی ترافیک تولیدی توسط آن‌ها رمزگذاری شده‌است ناکارامد می‌شود.

در پژوهشی که توسط الشهری[۵۵] ارائه‌شده است تفکیک ترافیک به وسیله‌ی یک طبقه‌بند انجام می‌شود. پس از تفکیک ترافیک متغیرهای هر بسته شامل Request, Value, Get, Host مشخص می‌شود.

---

[۴۲] Network Traffic
[۴۳] Source
[۴۴] Traffic Stream Matching
[۴۵] Hungarian Matching

ترافیک‌های از نوع Request در این قسمت به دو نوع اجباری و یا غیراجباری تقسیم می‌شود.ترافیک غیر اجباری شامل ترافیک مصرفی برنامک برای ارتباط با واسطهای دسترسی عنوان شده‌است. جداسازی ترافیک‌های شبکه از آن جهت اهمیت دارد که بسیاری از کتاب‌خانه‌های رایگان و یا حتی بدافزارهای موجود، یک نقطه‌ی دسترسی[۴۶] توسط واسطهای برنامه‌نویسی برای دسترسی به کتابخانه‌ها ایجاد کرده‌اند که برنامک‌های اندرویدی به وفور از این نقاط دسترسی استفاده می‌کنند. از طرفی تشخیص و جداسازی ترافیک اجباری و غیر اجباری موجب کاهش خطای منفی غلط می‌شود. برای شباهت‌سنجی ترافیک اجباری شامل ترافیک اصلی برنامک، از یک الگوریتم جریان‌درخواست[۴۷] مبتنی بر فاصله‌ی اقلیدوسی[۴۸] استفاده شده‌است. پیشنهاد جداسازی ترافیک کتابخانه‌ای و اصلی به وسیله‌ی نقطه‌های دسترسی واسطهای برنامه‌نویسی ایده‌ای نو در این پژوهش است که هر چند کامل نمی‌تواند ترافیک شبکه را جداسازی کند اما در صورت تکامل می‌تواند دقیق‌تر عمل کند.

در پژوهش دیگری که توسط آقای هه[۵۶] ارائه شده، از یک طبقه‌بند به جهت تشخیص برنامک‌های بازبسته‌بندی شده استفاده شده است. در پژوهش اخیر، ابتدا تمامی ترافیک کاربران متصل به یک شبکه به یک کارگزار سطح فرستاده می‌شود و این کارگزار ترافیک شبکه را برچسب‌زنی[۴۹] کرده و پس از استخراج ویژگی‌هایی نظیر محتوای[۵۰] اطلاعات هر بسته، آن‌ها را برای محاسبه‌ی امضا، سمت یک کارگزار مرکزی که به صورت ابری[۵۱] خدمات ارائه می‌کند، می‌فرستد. پس از ارسال ویژگی‌های مستخرج به سمت کارگزار ابری، فرایند شباهت‌سنجی آغاز می‌گردد. به جهت حفظ کامل حریم خصوصی کاربران، تحلیل ترافیک تنها برروی ترافیک رمزگذاری نشده‌ی(http) انجام می‌شود. در سمت کارگزار، با استفاده از تحلیل ترافیک شبکه‌ی هر کاربر، جریان اطلاعات در ترافیک برچسب‌زده شده مشخص می‌شود و سپس با حذف قسمتی از ترافیک در هر جریان به جهت کاهش اندازه، مانند پاسخ درخواست‌ها[۵۲] و قسمتی از سربرگ[۵۳] ترافیک، طبقه‌بندی صورت می‌گیرد. برای طبقه‌بندی از الگوریتم پرسرعت پژوهش [۵۷] استفاده شده‌است.

مالک و همکاران[۵۸] در پژوهش دیگری تحت عنوان CREDROID از یک روش مبتنی بر درخواست نام دامنه[۵۴] برای تشخیص برنامک‌های بازبسته‌بندی شده استفاده کرده است.در این پژوهش ابتدا ترافیک کاربر برای تحلیل به یک کارگزار مورد اعتماد فرستاده شده و فرایند بررسی عمیق ترافیک مورد نظر آغاز می‌شود. در ادامه و در سمت کارگزار مورد اعتماد، ترافیک کاربر ارزیابی شده و درخواست‌های نام دامنه

[۴۶] EndPoint
[۴۷] Request Flow
[۴۸] Euclidian Method
[۴۹] Labeling
[۵۰] Packet Content
[۵۱] Cloud
[۵۲] Response
[۵۳] Header
[۵۴] Domain Name Requests

جداسازی و برای هر کاربر برچسب‌گذاری می‌شود. در قسمت شباهت‌سنجی ترافیک درخواستی، از یک روش مبتنی بر فاصله‌ی اقلیدوسی جهت بررسی متن درخواست‌های دامنه استفاده می‌شود و در نهایت برنامک‌های بازبسته‌بندی شده شناسایی می‌شوند.

از آن‌جایی که بازبسته‌بندی برنامک‌های اندرویدی روشی محبوب برای حمله‌کنندگان به جهت تزریق و گسترش بدافزارهای اندرویدی است، تمرکز پژوهش ایلند و همکاران [۵۹] بر روی برنامک‌های اندرویدی بازبسته‌بندی شده و حاوی ترافیک مشکوک به بدافزار، با توجه رفتار مدل‌شده بر اساس درخواست‌های دامنه می‌باشد. این روش، مبتنی بر ترافیک حاصل از درخواست نام دامنه توسط برنامک‌های بازبسته‌بندی شده و سپس ارتباط آن با آدرس به دست آمده از درخواست بوده است. سپس ترافیک فرستاده‌شده از سمت کاربر برای آدرس‌هایی که به دست آمده‌اند بازبینی شده و جمع‌آوری می‌شود و در ادامه با استفاده از روشی مبتنی بر تطابق رشته‌های[۵۵] درخواست، شباهت‌سنجی صورت گرفته و برنامک‌های بازبسته‌بندی شده مشخص می‌شوند. یکی از ایرادات این پژوهش آن است که مبتنی بر رفتار بدافزارهای بیشتر شناخته‌شده انجام شده‌است و در صورتی که بدافزاری رفتار مشابه با بدافزارهای محبوب نداشته باشد شناسایی نمی‌شود. همچنین تحلیل ترافیک شبکه‌ی کاربر در ترافیک خام و رمزگذاری‌نشده صورت می‌گیرد و در صورتی که ارتباطات بدافزار بازبسته‌بندی شده حاوی ترافیک رمزگذاری‌شده باشد، آن‌گاه روش پیشنهادی در تشخیص آن‌ها ناتوان خواهد بود.

استفاده از روش‌های ترکیب با تحلیل ترافیک شبکه نظیر بررسی دسترسی‌های کاربران، در پژوهش شارما و همکاران [۶۰] استفاده شده‌است. تشخیص برنامک‌های بازبسته‌بندی شده در این پژوهش مبتنی بر طبقه‌بندی برنامک‌ها با استفاده از تحلیل ترافیک شبکه و بررسی مجوزهای دسترسی مورد نیاز برنامک می‌باشد. پس از دیس‌اسمبل برنامک، مجوزهای دسترسی از فایل فراداده‌ی[۵۶] برنامک مورد نظر استخراج شده و تشکیل یک بردار ویژگی دودویی را می‌دهند. این پژوهش از دو سطح برنامک‌های مورد نظر را بررسی می‌کند، در سطح اول اگر بردار ویژگی برنامک با برداری از برنامک‌های بازبسته‌بندی شده تطابق داشته باشد، آن‌گاه برنامک مورد نظر به عنوان یک برنامک مشکوک به سطح بعد فرستاده می‌شود. در سطح دوم، ترافیک رمزگذاری‌نشده‌ی برنامک تفکیک شده و تحلیل روی ترافیک TCP ادامه می‌یابد، سپس با استفاده از یک طبقه‌بند درخت تصمیم[۵۷]، برنامک‌های بازبسته‌بندی شده با استفاده از تحلیل ترافیک کاربران،مشخص می‌شوند.

اگر چه پژوهش‌های صورت‌گرفته در این دسته معمولا سرعت خوبی از نظر مقایسه‌ی مدل استخراج شده دارند، اما باید توجه داشت که محدودیت‌های موجود در این قسمت مانند به خطر افتادن حریم خصوصی

[۵۵] String Matching
[۵۶] Manifest File
[۵۷] Decision Tree

کاربران، از طریق بررسی ترافیک رمزنگاری شده، یکی از ویژگی‌های منفی پژوهش‌های مبتنی بر ترافیک برنامک می‌باشد. از طرفی، بسیاری از برنامک‌های اندرویدی، به صورت برون‌خط فعالیت می‌کنند که موجب می‌شود روش‌های این دسته اساساً از بررسی آن‌ها ناکام بمانند.

روش‌های مبتنی بر تحلیل منابع

عمده‌ی روش‌های موجود در این دسته، فرض استفاده از منابع برنامک‌های بازبسته‌بندی شده را در تعریف بازبسته‌بندی اعمال کرده‌اند. در این دسته از تعریف بازبسته‌بندی، متقلب با تغییر کدهای برنامک و استفاده‌ی مستقیم از منابع آن، سعی در جعل برنامک اصلی دارد. در این حالت رابط کاربری برنامک مشابه با رابط کاربری برنامک اصلی است اما کارکرد منطقی آن دجار تغییرات زیادی می‌شود بنابراین شناسایی این دسته از برنامک‌ها با استفاده از ویژگی‌های مبتنی بر منابع آن انجام می‌شود. به عنوان اولین پژوهش از این دسته، شائو و همکاران[۱]، برای تشکیل امضا هر برنامک، از دو دسته ویژگی آماری[۵۸] و ساختاری[۵۹] استفاده می‌کنند. در قسمت ویژگی‌های آماری، ۱۵ ویژگی شاخص از منابع برنامک‌های اندرویدی نظیر تعداد فعالیت‌ها[۶۰]، تعداد مجوزهای دسترسی، تعداد فیلترهای تصمیم[۶۱] و میانگین تعداد فایل‌های png و xml استخراج می‌شود. علاوه بر این ۵ ویژگی، ۱۰ ویژگی دیگر آماری شامل میانگین فراخوانی به ۱۰ منبع پر بازدید در برنامک‌های اندرویدی،مطابق با جدول ۳ـ۱ ، استخراج می‌شود.

پس از استخراج ۱۵ ویژگی آماری، دو ویژگی ساختاری دیگر مبتنی بر لایه‌ی فعالیت و کنترل‌گر رویدادهای[۶۲] برنامک ساخته می‌شود. برای ساخت این دو ویژگی از یک درخت انتزاعی مبتنی بر فعالیت‌های برنامک، استفاده شده‌است. در قسمت شباهت‌سنجی، پس از نرمال‌سازی ویژگی‌های استخراج شده در مراحل قبل، از دو روش طبقه‌بندی مبتنی بر نزدیک‌ترین همسایه و طبقه‌بندی طیفی[۶۳] برای تشخیص برنامک‌های تقلبی استفاده شده‌است.

---

[۵۸] Statistical
[۵۹] Structural
[۶۰] Activities
[۶۱] Intent Filters
[۶۲] Event Handler
[۶۳] Spectral Classification

جدول ۳ـ۱: ۱۰ منبع پربازدید در پژوهش [۱]

#	نوع منبع
1	id
2	drawable
3	string
4	color
5	style
6	dimen
7	layout
8	xml
9	integer
10	array

در پژوهش دیگری که توسط لین و همکاران[۶۱] انجام شده‌است، تشخیص بازبسته‌بندی با استفاده از چکیده‌ی تصاویر موجود در فایل‌های apk موجود در پوشه‌ی منابع، صورت می‌گیرد. در این پژوهش ابتدا چکیده‌ی تمامی تصاویر استخراج شده و شباهت‌سنجی میان آن‌ها به صورت تجمعی صورت می‌گیرد. در نهایت برنامک‌هایی که امتیاز تشابه آن‌ها از یک حد آستانه بیشتر باشد به عنوان برنامک‌های بازبسته‌بندی شده تشخیص داده می‌شوند. یکی از مفروضات پژوهش، به جهت کاهش فضای مقایسه، عدم لو رفتن کلید خصوصی توسعه‌دهنده‌ی برنامک می‌باشد. علاوه بر این، استفاده از روش‌های چکیده‌سازی معمول در این پژوهش، منجر به مقاومت پایین آن در مقابل مبهم‌نگاری می‌شود.

سان و همکاران[۶۲] با استفاده از تحلیل گراف فعالیت‌های برنامک،مجوعه‌ای از تصاویر ضبط‌شده[۶۴] هر برنامک را تحت هر فعالیت استخراج کرده و هر دید[۶۵] موجود در تصاویر را با استفاده از یک مستطیل نشان‌دهی و در نهایت به ازای هر تصویر، یک گروه از ناحیه دیدها ساخته می‌شود. در نهایت با استفاده از ادغام نواحی و گراف فعالیت‌های برنامک، گرافی تحت عنوان گراف گروه‌ـ‌ناحیه ساخته می‌شود. برای مقایسه و شباهت‌سنجی، با استفاده از پیمایش گراف‌ها، گره‌های گراف به عنوان نواحی برنامک با یکدیگر مقایسه می‌شود. دو ناحیه در صورتی با یکدیگر یکسان هستند که تعدادی از مجموعه‌های مدل‌شده به شکل مستطیل در این نواحی با یکدیگر هم‌پوشانی داشته باشند.

هو و همکاران[۶۳] با بررسی برنامک‌های بازبسته‌بندی شده متوجه شدند که ویژگی‌های ساختاری واسط‌های

[۶۴] ScreenShot
[۶۵] View

کاربری[۶۶] همانند طول و اندازه‌ی اشکال و دکمه‌ها کم‌تر دچار تغییر می‌شود و بیشتر ویژگی‌های محتوایی مانند رنگ پس‌زمینه تغییر می‌کند. با توجه به این یافته، آن‌ها سه ویژگی مهم مبتنی بر ساختار واسط‌های کاربری را استخراج و تحت عنوان امضای برنامک استفاده می‌کنند و در ادامه، این سه ویژگی را به یک فعالیت نسبت می‌دهند. در نهایت با کنار هم قرار گرفتن سه‌تایی‌های هر فعالیت و مقایسه‌ی آن‌ها با یکدیگر برنامک‌های تقلبی مشخص می‌شوند.

هدف آقای لین در پژوهش [۶۴] کاهش نرخ منفی‌غلط و مثبت‌غلط بوده‌است. در این پژوهش از ساختار سلسله‌مراتبی[۶۷] فایل‌های موجود در یک apk استفاده شده‌است. ساختار سلسله مراتبی مبتنی بر فایل‌های برنامک استخراج می‌شود و گره‌های این درخت برچسب‌گذاری می‌شود. برچسب‌گذاری درخت در این پژوهش، مبتنی بر نوع فایل مورد نظر آن گره انجام می‌شود چرا که برچسب‌گذاری بر اساس نام فایل‌ها، مقاومت پایینی در مقابل مبهم‌نگاری دارد و متقلب می‌تواند به راحتی روش را دور بزند. علاوه بر این مشکل دیگری که در این پژوهش وجود دارد، این است که در صورتی که چندین فایل از یک نوع وجود داشته باشد، گره‌های مشابه در درخت سلسله‌مراتبی به وجود می‌آید و در نهایت نرخ مثبت‌غلط به شدت افزایش پیدا می‌کند. برای حل این مشکل، نویسنده از یک ساختار چند مرحله‌ای برای تشخیص برنامک‌های بازبسته‌بندی شده استفاده می‌کند. در این ساختار ابتدا برچسب‌گذاری به نحوی انجام می‌شود که نرخ مثبت غلط زیادی در نتایج وجود داشته باشد و در ادامه از روشی با نرخ منفی غلط بالا استفاده می‌کند تا دقت پژوهش را افزایش دهد. شباهت‌سنجی درخت‌ها با استفاده از فاصله ویرایشی میان دو درخت انجام می‌شود. روش مورد نظر مقاومت پایینی در مقابل مبهم‌نگاری‌های ساده به خصوص افزونگی فایل‌ها، خواهد داشت چرا که متقلب می‌توان به راحتی با ایجاد تعداد زیادی فایل بلااستفاده، سیستم تشخیص را دور بزند.

گوآ و همکاران[۶۵] با استفاده از طرح‌بندی[۶۸] و ایجاد درخت مبتنی بر آن برنامک‌های بازبسته‌بندی شده را تشخیص می‌دهند. ابتدا درختی از تمامی مولفه‌های از نوع منبع و با استفاده از فایل‌های xml، به نام درخت طرح‌بندی کامل[۶۹]، استخراج می‌شود. درخت مذکور حاوی تمامی مولفه‌های طرح‌بندی یک برنامک اندرویدی است که تنها یک ریشه دارد. در ادامه برای مقایسه و شباهت‌سنجی درخت طرح‌بندی کامل، از یک روش مبتنی بر چکیده‌سازی $CTPH$ استفاده شده‌است.

استفاده از طرح‌بندی در پژوهش‌های مبتنی بر منابع محبوبیت زیادی دارد. برای مثال، در روش لیو و همکاران [۶۶]، ابتدا طرح‌بندی‌های مربوط به کتابخانه‌های اندرویدی حذف شده، سپس هر طرح‌بندی به یک درخت نگاشت می‌شود. به جهت مقایسه، طرح‌بندی‌های برنامک با یکدیگر ادغام می‌شوند و درخت نهایی در قالب یک فایل xml کدگذاری شده و در ادامه با استفاده از مقایسه‌ی مبتنی بر چکیده‌سازی فازی

[۶۶] User Interface
[۶۷] File Structure
[۶۸] Layout
[۶۹] Total tree layout

فایل‌های xml، برنامک‌های بازبسته‌بندی شده مشخص می‌شوند. در نمونه‌ی دیگری، لیو و همکاران[۶۷] در یک ساختار دو مرحله‌ای مبتنی بر طرح‌بندی ، برنامک‌های بازبسته‌بندی شده را تشخیص می‌دهند. مرحله‌ی اول در این پژوهش شامل تشخیص برنامک‌های مشکوک به صورت درشت‌دانه[۷۰] است که با استفاده از تصاویر موجود در پوشه‌ی منابع انجام می‌گیرد. در ادامه و به صورت ریزدانه، درخت طرح‌بندی تشکیل شده و امضای هر برنامک را تشکیل می‌دهد. برای مقایسه درخت‌ها، از روش‌های مرسوم مانند فاصله ویرایشی، استفاده شده‌است.

به عنوان یکی از آخرین پژوهش‌های موجود در این دسته به جهت بهبود روش‌های [۶۸، ۶۲] ما و همکاران[۶۹] با معرفی گراف انتزاعی طرح‌بندی[۷۱] و گراف طرح‌بندی انتقالی[۷۲]، برنامک‌های بازبسته‌بندی شده را شناسایی می‌کنند. در این پژوهش در ابتدا طرح‌بندی‌های هر برنامک تحت عنوان یک گراف جهت‌دار مدل می‌شود که هر گره گراف مذکور شامل طرح‌بندی‌های مشابه می‌باشد. در مقایسه با [۶۸]، روش ما و همکاران، از سرعت بیشتری در کدگذاری طرح‌بندی و تبدیل آن‌ها به گراف انتزاعی برخوردار است. علاوه بر این در قسمت شباهت‌سنجی و مقایسه، از روشی بهبودیافته، مبتنی بر یافتن تطبیق گراف بیشینه[۷۳] استفاده شده که ارزیابی عملکر پژوهش نسبت به دو پژوهش مورد مقایسه، افزایش سرعت و دقت را به همراه داشته است.

۳ـ۳ـ۲ مبتنی بر تحلیل پویا

به صورت کلی پژوهش‌های صورت‌گرفته در این قسمت را می‌توان به دو بخش روش‌های مبتنی بر جعبه‌شن[۷۴] و یا روش‌های خودکار تقسیم‌بندی کرد. در روش‌های مبتنی بر جعبه‌شن، پژوهش‌کنندگان با ایجاد محیطی شبیه‌سازی شده و تعامل حداکثری با برنامک اندوریدی مدنظر، سعی در شبیه‌سازی رفتار برنامک و جمع‌آوری مجموعه‌ای از ویژگی‌های موردنیاز به جهت مقایسه با یکدیگر دارند. این دسته از روش‌های پویا دو ایراد اساسی دارند، نخست آن‌که ایجاد یک محیط شبیه‌سازی شده و اجرای هر برنامک به طوری که تمامی رفتارهای آن‌را شبیه‌سازی کند، نیازمند زمان زیادی است که ممکن است روش را ناکارامد کرده و بررسی برنامک‌های موجود را محدود نماید. علاوه بر این ایجاد محیطی که بتواند به صورتی کامل رفتار برنامک را شبیه‌سازی کند نیازمند آن‌است که به نوعی تمامی خدمات هر برنامک منحصرا بررسی شود که این عملی تقریبا ناممکن است مگر آن‌که شبیه‌سازی منحصر یک برنامک خاص پیاده‌سازی شود. ایراد دوم آن است که برنامک‌های بازبسته‌بندی شده که دارای بدافزار هستند، ممکن است پیاده‌سازی آن‌ها قادر به شناسایی محیط

[۷۰] Coars Grain
[۷۱] Abstract Layout Graph
[۷۲] Abstract Transition Graph
[۷۳] Maximum Graph Matching
[۷۴] Sand Box

شبیه‌سازی شده باشد و در نتیجه، از اجرای قسمت‌هایی از برنامک خودداری کرده و سیستم تشخیص را به خطا بیندازد. برخلاف روش‌های مبتنی بر جعبه‌شن، روش‌های مبتنی بر تحلیل خودکار بیشتر سعی در استفاده از داده‌های کاربران واقعی برنامک‌های اندرویدی دارند. بیشتر روش‌ها در این دسته از معماری‌های ابری و کارخواه‌ـ کارگزار استفاده کرده و با استفاده از ارسال گزارشات[۷۵] از سمت کاربران شباهت‌سنجی را انجام می‌دهند. چندین نمونه از این روش‌ها را مانند [۳، ۴۳] در قسمت‌های پیشین بررسی کردیم. روش‌های مبتنی بر تحلیل خودکار هیچ شبیه‌سازی در مورد برنامک انجام نمی‌دهند و رفتار برنامک را با توجه به رفتار کاربران مورد تحلیل قرار می‌دهند. تمرکز ما در این قسمت روی پژوهش‌هایی است که بیشتر تمرکز آن‌ها بر پیاده‌سازی روشی مبتنی بر جعبه‌شن و شبیه‌سازی رفتار برنامک‌ها بوده‌است.

در پژوهش والریو و همکاران[۷۰] با استفاده از اجرای برنامک‌های اندرویدی در چندین جعبه‌شن مشهور و جمع‌آوری اطلاعات برای هر برنامک اندرویدی، یک پروفایل‌ـ استفاده[۷۶] ساخته می‌شود. این پروفایل نشان‌دهنده‌ی استفاده‌ی برنامک از ۸ خدمت مبتنی بر منابع سیستمی شامل لیست تماس‌ها، موقعیت مکانی، پیامک[۷۷]، شبکه‌ی ارتباطی بی‌سیم[۷۸]، برنامک‌ها، باتری و تماس‌ها می‌باشد. روند کلی جمع‌آوری امضای برنامک به این شکل است که برنامک‌های موجود در مخزن توسط یک مولفه در چندین جعبه‌شن مشهور و در دسترس اجرا شده و ویژگی‌های ذکر شده استخراج می‌شود و در نهایت تشکیل یک بردار ویژگی می‌دهند. در مرحله‌ی انتهایی، بردارهای ویژگی برنامک‌های اندرویدی با یکدیگر مقایسه شده و بسته‌های بازبسته‌بندی شده مشخص خواهند شد.

از آن‌جا که بررسی پردازش برنامک‌های اندرویدی به ازای ورودی‌های یکسان می‌تواند نتایج مشابه در برنامک‌های اندرویدی داشته باشد، ژانگ و همکاران[۷۱] روشی را مبتنی بر زمان مورد نیاز برای اجرای ورودی‌های یکسان توسط پردازنده[۷۹]، پیاده‌سازی کرده‌اند. در این پژوهش ، ابتدا تعدادی ورودی به هر برنامک اندرویدی موجود در مخزن برای اجرا داده می‌شود و پس از اتمام پردازش، بازه‌ی زمانی اجرای پردازه در پردازنده ضبط می‌شود و تحت عنوان یک دوتایی <پردازه، زمان> نمایش داده می‌شود. ورودی‌های یکسان برای تمامی برنامک‌های مورد بررسی اجرا می‌شود و در نهایت لیستی از دوتایی‌های موجود برای هر برنامک، تشکیل امضای آن‌را می‌دهد.

در روش دیگری که توسط نگویان و همکاران[۷۲] اجرا شده، برنامک‌های اندرویدی با یک روش مبتنی بر جعبه‌شن و استخراج ویژگی‌هایی از واسطکاربری برنامک، بسته‌های بازبسته‌بندی شده را شناسایی می‌کنند. در این پژوهش با استفاده از یک جعبه‌شن که وظیفه‌ی آن ضبط تصاویر محیط کاربر برنامک می‌باشد،

[۷۵] Logs
[۷۶] Usage Profile
[۷۷] SMS
[۷۸] Wifi
[۷۹] CPU

تصاویری از برنامک‌های اندرویدی مورد بررسی ذخیره می‌شود. از آن‌جایی که روش‌های چکیده‌سازی مرسوم، چکیده‌ای کاملا متفاوت حتی برای کوچکترین تغییرات تولید می‌کنند، در این پژوهش از روشی مبتنی بر چکیده‌سازی ادراکی[۸۰] استفاده شده‌است. در این دسته از روش‌های چکیده‌سازی، تغییرات کوچک به همان اندازه باعث تغییرات کوچک در چکیده‌ی تولیدشده می‌شود و فایل‌های مشابه اما با تغییرات اندک می‌توانند پس از مقایسه‌ی چکیده‌سازی ادراکی، شناسایی شوند. در مرحله‌ی شباهت‌سنجی، تصاویر ضبط‌شده از هر برنامک، با استفاده از روش‌های میانگین چکیده‌سازی[۸۱]، با تصاویر برنامه‌های موجود در مخزن مورد مقایسه قرار می‌گیرند و در نهایت برنامک‌های بازبسته‌بندی شده به این طریق تشخیص داده می‌شوند.

در پژوهش دیگری که مبتنی بر واسط‌های کاربری انجام شده‌است،سو و همکاران[۷۳] با استفاده از استخراج فعالیت‌های برنامک موجود در فایل فراداده‌ی آن‌ها، فعالیت‌های مورد نظر را در یک محیط مبتنی بر جعبه‌شن اجرا و تصاویر حاصل از اجرای هر فعالیت را ضبط می‌کنند. در قسمت شباهت‌سنجی، اطلاعاتی از درون هر کدام از تصاویر ضبط‌شده از برنامک حین اجرای آن، استخراج می‌شود و تشکیل بردار ویژگی هر برنامک را می‌دهند. در ادامه با استفاده از روش‌های مبتنی بر چکیده‌سازی محلی[۸۲]، شباهت بردارهای ویژگی با یکدیگر سنجیده و در صورتی که میزان شباهت از یک حد آستانه بیشتر باشد، جفت مورد بررسی، بازبسته‌بندی‌شده یکدیگر شناسایی می‌شوند. برنامک برای مقاومت بیشتر در مقابل مبهم‌نگاری از لیست فعالیت‌های برنامک استفاده کرده و نقاط ورود[۸۳] را از روی این لیست فعالیت مشخص می‌کند. این پژوهش، فعالیت‌های موجود در برنامک‌های اندرویدی را مستقل از یکدیگر در نظر گرفته‌است و به همین دلیل نقاط ورود متنوعی مبتنی بر هر فعالیت تعیین می‌شود، در صورتی که در اکثر برنامک‌های اندرویدی، فعالیت‌های موجود کاملا به هم وابسته هستند و گاهاً یک فعالیت نتیجه‌ی یک فعالیت دیگر است بنابراین فرض پژوهش در این قسمت به نظر اشتباه می‌باشد.

به طور کلی، تشخیص برنامک‌های اندرویدی مبتنی بر روش‌های پویا شامل استخراج ویژگی‌های متنوع ساختاری و رفتاری برنامک در حین اجرای آن در محیط شبیه‌سازی شده است. برای مثال، در پژوهش بلاسینگ و همکاران [۷۴] با استفاده از یک برنامک تزریق‌شده به هسته[۸۴]ی سیستم‌عامل، تمامی فراخوانی‌های سیستمی ناشی از اجرای برنامک در محیط جعبه‌شن، استخراج می‌شود. استخراج فراخوانی سیستمی از طریق هسته‌ی سیستم‌عامل، منجر به مقاومت بالای این روش مقابل مبهم‌نگاری می‌شود. کیم و همکاران [۷۵]، برای تشخیص برنامک‌های بازبسته‌بندی شده، از بردارهای ویژگی مبتنی بر فراخوانی‌های

[۸۰] Perception Hashing-Phash
[۸۱] Average Hashing
[۸۲] Locality Sensitive Hash
[۸۳] Entry Point
[۸۴] Kernel

واسط‌های برنامه‌نویسی[۸۵] استفاده کرده‌اند و در نهایت با مقایسه‌ی برنامک‌ها، مسئله را حل می‌کنند. گوان و همکاران[۷۶]، با استفاده از یک روش مبتنی بر معناشناسی[۸۶] در برنامک‌های اندرویدی، بسته‌های بازبسته‌بندی شده را شناسایی می‌کنند. آن‌ها با بررسی مشکلات تشخیص در تحلیل‌های ایستا، در ابتدا گراف جریان ارتباطی میان کلاس‌های برنامک را استخراج کرده و در ادامه با استفاده از وزن‌دهی گراف، مبتنی بر ورودی و خروجی کلاس، کلاس‌های اصلی برنامک را از کلاس‌های فرعی جدا می‌کنند. سپس متدهای هر کلاس با استفاده از گراف جریان میان متدهای کلاسی، به صورت نمادین اجرا می‌شوند و با بررسی تمامی جایگشت‌های ممکن به عنوان ورودی هر متد و خروجی آن‌ها، متدهای یکسان و شبیه به هم به دست می‌آیند. تشخیص بازبسته‌بندی با استفاده از یک حد آستانه، بر اساس تعداد متدهای یکسان مشخص می‌شود. یو و همکاران[۶۸] با استفاده از ساخت گراف فعالیت مبتنی بر واسط کاربری هر برنامک و در نهایت مقایسه‌ی گرافی، برنامک‌های بازبسته‌بندی شده را تشخیص داده‌اند. برای مقایسه‌ی گرافی، پس از وزن‌دهی گراف بر اساس تعداد تکرار هر فعالیت، از یک الگوریتم تطبیق بیشینه‌ی گراف، استفاده شده‌است.

۳ـ۳ـ۳ سایر روش‌ها

وانگ و همکاران[۷۷] از روشی مبتنی بر طبقه‌بندی برای شناسایی کتابخانه‌های اندرویدی استفاده کرده‌اند. با توجه به این که کتابخانه‌های اندرویدی در برنامک‌های مختلف تکرار می‌شوند بنابراین جمع‌آوری دنباله‌ی فراخوانی واسط‌های برنامه‌نویسی در کتاب‌خانه‌های مشابه، منجر به ساخته‌شدن دنباله‌های مشابه می‌شود. بنابراین پس از جمع‌آوری دنباله‌ی فراخوانی‌ها از برنامک‌های متعدد، طبقه‌بندی روی دنباله‌ی ورودی با استفاده از داده‌ی آموزشی انجام می‌شود و در نهایت کدهای کتابخانه‌ای با استفاده از این روش جدا می‌شوند. در قسمت دوم از یک راه حل دو مرحله‌ای برای تشخیص برنامک‌های اندرویدی بازبسته‌بندی‌شده استفاده شده‌است. پس از حذف کدهای کتابخانه‌ای، دنباله‌های فراخوانی واسط‌های اندرویدی در هر برنامک محاسبه و دنباله‌های موجود با یکدیگر مقایسه می‌شوند. در این مرحله، برنامک‌های مشکوک به بازبسته‌بندی تشخیص داده می‌شوند و در مرحله‌ی بعدی بررسی دقیق‌تر صورت می‌گیرد. در مرحله‌ی دوم، متغیرهای موجود در برنامک در روشی مبتنی بر توکن[۸۷]، به صورت برداری نشان داده می‌شوند. به جهت شباهت‌سنجی و مقایسه‌ی بردارهای موجود، تعداد بردارهای یکسان، که مبتنی بر شناسه‌ی هر متغیر و نام آن است، با یکدیگر مقایسه می‌شود و در صورتی که تعداد توکن‌های مشابه، در یک جفت برنامک از یک حد آستانه بیشتر باشد، جفت مورد بررسی به عنوان بازبسته‌بندی در نظر گرفته می‌شوند. روش پژوهش، از آن‌جا که در دومرحله برنامک‌های بازبسته‌بندی شده را شناسایی می‌کند از سرعت خوبی برخوردار است، اما با استفاده

---

[۸۵] API Calls
[۸۶] Semantic
[۸۷] Token

از روش‌های مبتنی بر افزودن متغیرهای بیهوده، می‌توان روش یادشده را دور زد.

نیشا و همکاران[۷۸]، با استفاده از ویژگی‌های مبتنی بر مجوزهای دسترسی و ایجاد بردار دودویی مجوزهای هر برنامک، به وسیله‌ی روش‌های یادگیری ماشین برنامک‌های بازبسته‌بندی شده را تشخیص می‌دهند. یکی از مشکلات روش موجود آن است که برای ایجاد مجموعه‌ی داده آزمون، از یک روش مبتنی بر کپی مجدد برنامک‌های اصلی استفاده شده‌است که باعث می‌شود روش موجود در عین داشتن دقت بالا در مورد مجموعه‌ی آزمون پژوهش، به صورت کلی مقاومت بالایی در تشخیص بازبسته‌بندی نداشته باشد. مشکل دیگر این روش، آن است که عملا تعریف تشخیص بازبسته‌بندی در این روش‌ها، حل مسئله‌ی تصمیم در مورد بازبسته‌بندی است و شامل تشخیص جفت تقلبی نمی‌شود.

۳ـ۴ پیش‌گیری از بازبسته‌بندی

دیدگاه پژوهش‌های پیشین در زمینه‌ی تشخیص برنامک‌های بازبسته‌بندی شده، بررسی برنامک‌های مشکوک پس از بازبسته‌بندی توسط مهاجم بوده‌است اما عمده‌ی پژوهش‌های موجود به جهت پیش‌گیری از بازبسته‌بندی نیازمند ایجاد تغییراتی در برنامک پیش از انتشار آن به صورت عمومی می‌باشد. این روش‌ها، از بازبسته‌بندی برنامک توسط متقلبان جلوگیری کرده و یا در صورتی که بازبسته‌بندی صورت بگیرد توسعه‌دهندگان متوجه آن خواهند شد. مزیت بزرگ روش‌های پیش‌گیری از بازبسته‌بندی آن است که از قطعیت بیشتری برخوردار هستند، یعنی در صورتی که بازبسته‌بندی صورت بگیرد اکثراً به صورت قطعی توانایی تشخیص و جلوگیری از انتشار برنامک‌ها را دارند. به صورت کلی پژوهش‌های اخیر در این حوزه را می‌توان به دو دسته تقسیم کرد که در ادامه به توضیح مختصری از هرکدام و شرح پژوهش‌های هر دسته می‌پردازیم.

۳ـ۴ـ۱ روش‌های مبتنی بر نشان‌گذاری

در این دسته از روش‌ها که نیازمند ایجاد تغییراتی در کد برنامک توسعه‌داده‌شده است، توسعه‌دهندگان قسمتی از امضای خود را تحت عنوان حق‌تکثیر[۸۸] که با نام نشان شناسایی می‌شود، در برنامک پنهان می‌سازند و در صورتی که برنامک جدیدی بازبسته‌بندی شود با توجه به تغییر این حق‌تکثیر قادر به تشخیص و جلوگیری از بازبسته‌بندی خواهند بود. به صورت کلی روش‌های نشان‌گذاری[۸۹] به دو نوع ایستا و پویا تقسیم می‌شوند. در روش‌های ایستا، پیش از ساخته‌شدن برنامک و انتشار آن، حق‌تکثیر ایجاد و در ساختار داده‌ای و یا کدهای برنامک تزریق می‌شود اما در روش‌های پویا، به جهت غلبه بر محدودیت روش‌های

[۸۸] Copy Right
[۸۹] Watermarking

ایستا و بررسی دقیق‌تر، حق‌تکثیر توسعه‌دهندگان در حین اجرای برنامک توسط کدهای مورد توسعه ایجاد می‌شود و در ساختار آن قرار می‌گیرد. در روش‌های مبتنی بر نشانگذاری از یک روند مشخص برای شناسایی و تایید نشان تزریق‌شده به برنامک‌ها استفاده می‌شود که عدم بازبسته‌بندی را پیش از اجرا و یا در حین اجرا، بسته به ایستا و یا پویا بودن روش، مشخص می‌کند. از طرفی به دلیل عدم قطعیت تشخیص در عمده‌ی روش‌های تشخیص، روش‌های نشانگذاری و روش‌های تشخیص برنامک‌های بازبسته‌بندی شده می‌توانند به صورت مکمل یکدیگر عمل کنند. در این حالت ابتدا برنامک‌های مشکوک با استفاده از روش‌های بیان‌شده در حوزه‌ی تشخیص، شناسایی می‌شوند و در ادامه به جهت اطمینان از صحت برنامک مورد نظر، توسط روش‌های مبتنی بر نشانگذاری، حق‌تکثیر توسعه‌دهندگان بررسی می‌شود و در صورتی که برنامک مدنظر بازبسته‌بندی شده باشد، از فروشگاه حذف خواهد شد. استفاده از هر دوی این روش‌های سرعت و دقت تشخیص برنامک‌های اندرویدی بازبسته‌بندی شده را افزایش می‌دهد.

یکی از اولین پژوهش‌های این حوزه توسط وو وهمکاران[۷۹] انجام شده‌است. در این پژوهش ابتدا رشته‌ای از اعداد و حروف برای تولید نشان مشخص می‌شود. سپس رقم مشخص‌شده در فرایندی مشخص به یک گراف جایگشت نگاشت شده و در ادامه گراف مورد نظر با استفاده از یک قطعه‌کد نشان‌داده می‌شود. برای جلوگیری از مبهم‌نگاری نشان در فرایند بازبسته‌بندی، کد ایجادشده برای گراف نشانگذاری، در قسمت‌های مختلف برنامک هدف تقسیم می‌شود. برای تایید برنامک اصلی و جلوگیری از بازبسته‌بندی از یک برنامک ثانویه که حکم فراداده‌ی برنامک اصلی را دارد استفاده می‌شود. برای ساخت برنامک ثانویه، از نمونه‌ی آزمونی که تمامی کد را پوشش دهد استفاده می‌شود هر ورودی آزمون در این نمونه، یک گرداننده دارد که هر تکه از کد نشانگذاری شده در قسمت قبل درون آن قرار می‌گیرد. در واقع هدف از تولید برنامک ثانویه، نگه‌داشت مکان‌هایی از برنامک اصلی است که کد نشانگذاری در آن قسمت‌ها قرار گرفته‌است چون همانطور که اشاره کردیم جهت جلوگیری از مبهم‌نگاری نشانگذاری در حین بازبسته‌بندی، کدهای نشانگذاری در قسمت‌های مختلف برنامک اصلی تقسیم می‌شوند. بنابراین در صورتی که برنامک ثانویه، در دسترس متقلب قرار گیرد، می‌تواند تمامی نشانگذاری‌های موجود در کد را شناسایی و آن‌ها را حذف کند بنابراین توسعه‌دهندگان نمی‌توانند از بازبسته‌بندی جلوگیری کنند.

ژانگ و همکاران[۸۰]، از یک روش نشانگذاری مبتنی بر تصاویر نشانگذاری‌شده استفاده می‌کنند. نویسندگان، با توجه به این که در پژوهش[۷۹] هیچ وابستگی داده‌ای میان کدهای نشانگذاری شده وجود ندارد، روشی را پیشنهاد داده‌اند تا مهاجم نتواند با تحلیل برنامک و یافتن کدهای نشانگذاری شده، در آن‌ها مبهم‌نگاری ایجاد کرده و سیستم تشخیص را دور بزند. در این روش، با استفاده از کاراکترهای اسکی[۹۰]، تصویر ورودی را کدگذاری کرده و طی یک دنباله‌ی تصادفی از رخدادها، کد تولیدشده‌ی اسکی در مسیر

[۹۰] Ascii

رخدادهای تولیدشده تقسیم می‌شود. به جهت این‌که از تغییر نشان‌ها توسط مهاجم جلوگیری شود، تعدادی از کاراکترهای تصویر ورودی را با اعداد ثابت موجود در کد برنامک جایگزین می‌سازند که موجب می‌شود تصویر ورودی تغییر چندانی نکند و در عین حال وابستگی داده‌ای میان کدهای برنامک و تصویر ورودی ایجاد شود. یکی از مشکلات دو روش اخیر، عدم امکان بررسی اصالت برنامک توسط کاربران می‌باشد. برای رفع این مشکل رن وهمکاران[۸۱] از کدگذاری بلوکی مبتنی بر ساختارهای خودرمزگشایی[۹۱] استفاده کرده‌اند. در این پژوهش، کد نشان‌گذاری شده پس از تولید، در بلوک‌های شرطی کد برنامک مورد توسعه قرار گرفته و رمزگذاری صورت می‌گیرد و در حین اجرا رمزگشایی بلوک‌های اجرایی منجر به تایید اصالت کدنشان‌گذاری شده می‌شود.

یکی از مشکلات روش‌های مطرح‌شده در این زمینه آن‌است که جلوگیری از بازبسته‌بندی همواره باید توسط توسعه‌دهندگان صورت گیرد. به همین جهت ایجاد روندی خودکار جهت جلوگیری از اجرای برنامک‌های بازبسته‌بندی شده توسط خود برنامک ایده‌ای بود که در پژوهش[۸۲] توسط سان و همکاران دنبال شده‌است. در این پژوهش ابتدا تمامی ساختارهای شرطی[۹۲] برنامک استخراج می‌شود و با استفاده از تحلیل و تغییر شروط به صورتی که منطق برنامک تغییر نکند، کد نشان‌گذاری شده در ساختار برنامک جایگذاری می‌شود. در نهایت با استفاده از یک جریان مبتنی بر بارگذاری پویا، در صورتی که کد نشان‌گذاری شده تغییری کرده‌باشد، از اجرای برنامک جلوگیری خواهد شد.

لوآ و همکاران[۸۳] در سال ۲۰۱۶ روشی را مبتنی بر کلید عمومی توسعه‌دهنده‌ی برنامک اندرویدی، برای جلوگیری از بازبسته‌بندی ارائه کرده‌اند. نویسندگان در طی روشی به نام شبکه‌ی پنهانی مرتبط[۹۳]، به عنوان ورودی، کد برنامک اندرویدی به همراه کلید عمومی کاربر توسعه‌دهنده‌ی برنامک را دریافت می‌کند و با استفاده مقایسه‌ی کلیدعمومی ثبت‌شده در داخل برنامک و زیررشته‌ای از محاسبات مبهم‌نگاری شده در برنامک هنگام اجرای آن، اصالت برنامک را تشخیص می‌دهند. برای به دست آوردن کلیدعمومی در حین اجرای برنامک، از ویژگی بازتاب در زبان جاوا استفاده شده‌است. یکی از مشکلات این پژوهش آن‌است که به دلیل استفاده از ویژگی بازتاب برای دستیابی به کلیدعمومی، حمله‌کننده می‌تواند قسمتی که متد فراخوانی می‌شود را بازنویسی کرده و از طریق ورودی‌های قبلی، متد را بازفراخوانی کند. در این صورت حمله‌کننده به راحتی می‌تواند به کلید عمومی دسترسی پیدا کرده و در ادامه با کمی تحلیل ایستا، قسمتی از برنامک را که کلیدعمومی در آن ثابت‌شده‌است را بیابد.به جهت مقابله با این نوع از حملات، زنگ [۸۴] راه‌حلی مبتنی بر بمب‌های منطقی[۹۴] ارائه کرده‌است. بمب‌های منطقی، قطعه کدهای کوچکی هستند که تنها در حالاتی خاص فعال می‌شوند. در طی این روش، کدهای نشان‌گذاری در ساختار شرط‌های منطقی

[۹۱] Self Decryption
[۹۲] Conditional Statement
[۹۳] Stochstic Stealthy Network
[۹۴] Logic Bomb

در برنامک تزریق می‌شود به صورتی که برخی از این شروط در طی اجرای برنامک در سیستم‌عامل کاربران معمولی اجرا می‌شود. در صورتی که مهاجم از روش‌های مبتنی بر جعبه‌سیاه استفاده ننماید، در آن صورت محیط اجرای وی محدود به تعدادی سیستم‌عامل و ویژگی‌های مبتنی بر آن خواهد بود، این در حالی است که به صورت کلی کاربران گستردگی بیشتری در تنوع این ویژگی‌ها دارند. بنابراین می‌توان شروط منطقی این بمب‌ها را به گونه‌ای نوشت، که کاربر مهاجم شانس کمی برای دسترسی به نشانگذاری داشته‌باشد. با استفاده از این روش، پژوهش [۸۳] توسعه یافت و تا حدودی مشکلات آن برطرف گردید، اما پژوهش جدید نیز دچار مشکلاتی شد که در ادامه به آن خواهیم پرداخت. در صورتی که کاربر بتواند با تحلیل پویا و فعال‌سازی شروط و در نهایت دسترسی به بلوک بمب‌های منطقی، پیش از اجرای کدهای رمزگشایی شده آن‌ها را تغییر دهد، آنگاه می‌تواند از تشخیص اصالت برنامک جلوگیری نماید. تانر و همکاران[۸۵] برای حل این مشکل از کدهای بومی استفاده کرده‌اند. از آن‌جایی که تغییر کدهای بومی در حین بازبسته‌بندی برای مهاجم دشوار و پرهزینه‌است، نویسندگان پس از دستیابی به کدهای بومی و کدهای اصلی برنامک، مجموعه‌ای از بایت‌کدهای دالویک رمزنگاری شده را در قسمت‌های مختلف برنامک اندرویدی اصلی تزریق کرده و متدهای بررسی صحت برنامک را در کدهای بومی پیاده‌سازی می‌کنند. رمزنگاری بایت‌کدها در کنار فراخوانی توابع بررسی صحت در کدهای بومی، موجب شده‌است که روش موجود، در مقابل تحلیل ایستا و پویا توسط مهاجم امنیت داشته باشد.

۳ـ۴ـ۲ روش‌های مبتنی بر مبهم‌نگاری

همانطور که در فصل ۲ اشاره‌شد، مبهم‌نگاری یکی از روش‌های پیش‌رو برای جلوگیری از بازبسته‌بندی توسط مهاجمان است. توسعه‌دهندگان از مبهم‌نگاری استفاده می‌کنند تا تغییردادن برنامک‌های اندرویدی خود را سخت و هزینه‌بر کنند. به صورت کلی مبهم‌نگاری در وهله‌ی اول موجب می‌شود تا ساختار منطقی اصلی و به طور مشخص‌تر، بدنه‌ای اصلی منطقی برنامک از دید مهاجمان مخفی بماند. در مرحله‌ی بعدی در صورتی که مهاجم بتواند بدنه‌ی منطقی اصلی برنامک اندرویدی قربانی را شناسایی کند، تغییردادن آن و سپس پیاده‌سازی منطق مهاجم به صورت یک بدافزار، کاری سخت، زمان‌بر و هزینه‌بردار خواهد بود، به طوری که ممکن است مهاجم از انجام این عمل امتناع ورزد. همانطور که در فصل پیش دیدیم، برخی از ابزارهای موجود، مبهم‌نگاری در برنامک‌های اندرویدی را انجام می‌دهند. اما پژوهش‌های موجود در این قسمت نیز، به صورت پیش‌رو، روش‌هایی را توسعه داده‌اند که می‌توانند نقش پیش‌گیری در بازبسته‌بندی برنامک‌های اندرویدی را داشته باشند.

یکی از روش‌های مبهم‌نگاری در برنامک‌های اندرویدی، تغییر نام شناسه‌ها، کلاس‌ها و متدهای برنامک می‌باشد. این روش موجب می‌شود تا تحلیل و جست‌وجوی رشته‌ها در برنامک دشوار شده و مهاجم نتواند

با تحلیل برنامک، آن را تغییر دهد. از آن‌جایی که تغییر نام در بیشتر مواقع تاثیری بر منطق برنامک ندارد، تغییر نام شناسه‌های برنامک، روشی است که ورمک و همکاران[۸۶] از آن استفاده کرده‌اند. پروگارد ابزار دیگری است که با تغییر نام کلاس‌ها، متدها و شناسه‌های موجود در برنامک مبهم‌نگاری را انجام می‌دهد. تغییر نام در پروگارد با استفاده از کاراکترهای محدود اسکی صورت می‌گیرد، به همین جهت دکس‌گارد معرفی شد تا نسخه‌ی پیشرفته‌تری از پروگارد را ارائه دهد که تغییر نام در آن‌ها با دامنه‌ی بیشتری از کاراکترهای اسکی انجام شود[۸۷].

برنامک‌های مبتنی بر زبان جاوا، با استفاده از پیشوندهای مشخص[۹۵] گروه‌بندی می‌شوند و در یک مسیر جاری قرار می‌گیرند که موجب می‌شود کلاس‌های موجود در یک گروه به صورت مستقیم به کلاس‌های هم‌گروهی خود دسترسی داشته باشند. تغییر نام بسته‌های اندرویدی، یکی از روش‌های معمول به جهت جلوگیری در بازبسته‌بندی است چرا که موجب می‌شود تحلیل برنامک برای مهاجم سخت شود. وانگ و همکاران[۸۸] از این روش برای افزایش مقاومت مقابل مبهم‌نگاری استفاده کرده‌اند. در روشی که توسط نویسندگان پیاده‌سازی شده‌است، تعدادی کلاس کمکی برای مخفی کردن کلاس‌های اصلی برنامک به بسته‌های نرم‌افزاری اضافه شده‌است. پژوهش[۸۷] نیز در تولید ابزار پروگارد از تغییر نام بسته‌ها استفاده کرده است.

حذف کدهای بومی‌ای که در هنگام اجرای برنامک نیازمند آن‌ها نیستیم، روش دیگری از مبهم‌نگاری برنامک‌های اندرویدی است که در پژوهش علم و همکاران [۸۹] از آن استفاده شده‌است. حذف کدهای بومی موجب می‌شود که خوانایی برنامک در حین دی‌کامپایل توسط مهاجمان کاهش یابد. علاوه بر این موضوع، به جهت خوانایی پایین کدهای بومی نسبت به بایت‌کدهای دالویک که یک زبان میانی محسوب می‌شود، در پژوهش علم و همکاران [۸۹]، نویسندگان قسمتی از برنامک را که شامل الگوریتم‌های رمزنگاری می‌شود، به صورت کدهای بومی پیاده‌سازی کرده‌اند که موجب می‌شود خوانایی این الگوریتم‌ها و در نتیجه دسترسی به الگوریتم‌های رمزگشایی سخت شود.

تغییر گراف کنترل جریان، روشی دیگری است که برای مبهم‌نگاری از آن استفاده می‌شود. تحلیل گراف کنترل جریان توسط مهاجمان، یکی از روش‌های محبوب بدافزارنویسان برای واردساختن بدافزار و بازبسته‌بندی است. گراف کنترل جریان نشان‌دهنده‌ی روال فراخوانی میان متدهای یک برنامک را نشان می‌دهد. باسی و همکاران[۹۰] با استفاده از افزودن یال به گراف جریان، آن را تغییر می‌دهند. افزودن یال با استفاده از اضافه کردن یک گره جدید در میان هر فراخوانی و ایجاد فراخوانی واسط انجام می‌شود. پردا و همکاران[۹۱] از همین روش برای تغییر گراف جریان استفاده کرده‌اند با این تفاوت که متد میانی اضافه‌شده، به صورت ایستا و با همان ورودی‌های متد اصلی تعریف می‌شود تا شناسایی متدهای اصلی

[۹۵] NameSpace

فراخواننده دشوار شود. افزودن کدهای بیهوده، روشی دیگری است که در پژوهش‌های[۹۲، ۹۳، ۹۴]، استفاده شده‌است. در پژوهش لی و همکاران[۹۲]، با استفاده از افزودن کدهای ،nop پرش‌های قطعی[۹۶] و ثبات‌های داده[۹۷] پردازش‌های بیهوده‌ای در برنامک اضافه می‌کنند که منطق برنامک را تغییر نمی‌دهد. راستوگی [۹۴]، حالتی از دستورات شرطی را معرفی کرد که همواره یک نتیجه می‌دهد اما گراف جریان را به دو شاخه تقسیم می‌کند. در این حالت یکی از حلقه‌های این دستور، شامل کد اصلی خواهد بود و حلقه‌ی دیگر شامل پرش بدون‌شرط به ابتدای کد اصلی. ژانگ [۹۳] در پژوهش خود از این روش برای مبهم‌نگاری برنامک‌های اندرویدی استفاده کرده‌است. جابه‌جایی دستورات، روش دیگری است که در پژوهش‌های [۸۷، ۹۲، ۹۱] استفاده شده‌است. باسی و همکاران[۹۲] با استفاده از جابه‌جایی دستورات به صورت کاملا تصادفی، با دانه[۹۸]های ناهم‌گون و در نهایت استفاده از دستورات پرش، خطوط کدهای اصلی را جابه‌جا کرده و به این شکل مهاجم را در تحلیل کدهای اصلی برنامک، ناکام می‌گذارد. پردا و همکاران[۹۱] با تحلیل دالویک بایت‌کدها، بلوک‌های مستقل اجرایی در آن‌ها را استخراج کرده و ترتیب آن‌ها را عوض می‌کنند. برای تضمین اجرای منطق اصلی برنامک، از شرط‌های منطقی قطعی استفاده شده‌است.

استفاده از رمزگذاری در حین کامپایل برنامک‌های اندرویدی و دسترسی به اطلاعات در حین اجرا، یکی دیگر از روش‌های مبهم‌نگاری است که در سال‌های اخیر محبوبیت زیادی پیدا کرده‌است. روش‌های رمزگذاری را به صورت کلی می‌توان به دو دسته‌ی رمزگذاری داده و رمزگذاری منابع برنامک تقسیم کرد. در روش‌های مبتنی بر رمزگذاری داده، قسمت‌های حساسی از کد برنامک، مانند کلید دسترسی به واسط‌های برنامه‌نویسی و یا گذرواژه‌های مهم مانند آنچه برای درخواست از پایگاه‌داده مورد استفاده قرار می‌گیرد، رمزگذاری می‌شود.باسی و همکاران[۹۰] با استفاده از این ایده و استفاده از رمزگذاری مبتنی بر سیستم رمز سزار[۹۹]، گذرواژه‌های حساس و حیاتی برنامک را رمزنگاری و در هنگام اجرا رمزگشایی می‌کند. منابع برنامک‌های اندرویدی، نیازمند کامپایل توسط ماشین مجازی جاوا نیستند و توسط کدهای برنامک، فراخوانی می‌شوند. پردا و همکاران[۹۱] با استفاده از چکیده‌سازی ۱۲۸ بیتی مبتنی بر MD5، فایل‌های منابع را چکیده و آن‌ها را در مسیر جدیدی مقداردهی می‌کند. از مشکلات این پژوهش آن‌است که نویسندگان به درستی، نحوه‌ی کارکرد کلاسی را که وظیفه‌ی بارگیری پویای فایل‌های منابع رمزشده را بر عهده دارد، مشخص نکردند. علاوه بر رمزگذاری منابع و اطلاعات حساس، رمزگذاری کلاس‌های برنامک نیز روش دیگری است که دیوید و همکاران[۹۵] از آن استفاده کرده‌اند. در این پژوهش، هر یک از کلاس‌های اندرویدی پس از رمزگذاری، در آرایه‌هایی از جنس بایت ذخیره می‌شود و کلاسی مجزا در هنگام اجرا، با استفاده از قابلیت بازتاب در برنامک‌های جاوا، کلاس رمزگذاری شده را بازیابی و اجرا می‌کند.

---

[۹۶] Unconditional Jump
[۹۷] Data Register
[۹۸] Seed
[۹۹] Caesar Cipher

به صورت کلی می‌توان تاثیر پژوهش‌های مبهم‌نگاری بر برنامک‌های اندرویدی را از سه دیدگاه بررسی کرد. در حالت اول مبهم‌نگاری‌هایی نظیر تغییرنام و یا اجرای کلاس‌های رمزگذاری‌شده، منجر به **افزایش امنیت و جلوگیری از بازبسته‌بندی** می‌شود. در حالت دوم، رمزگذاری داده‌ها و یا استفاده از کدهای بومی در هنگام اجرا، **مانع از اجرای تحلیل بدافزار** روی برنامک مورد نظر می‌شود. در حالت پایانی، روش‌های مبهم‌نگاری نظیر چکیده‌سازی، تغییر نام و یا حذف کدهای مرده و بیهوده، منجر به **افزایش کارایی** و بهینه‌شدن اجرای برنامک‌های اندرویدی می‌شود. از آن‌جایی که مهاجمان نیز از روش‌های مبهم‌نگاری استفاده می‌کنند، بنابراین آشنایی با پژوهش‌های اخیر موجود در این حوزه می‌تواند ما را در ارائه‌ی روشی با دقت بیشتر و سرعت بالاتر یاری دهد.

۳ـ۵ مقایسه‌ی روش‌ها

با توجه به تحلیل‌های ارایه‌شده بر روی پژوهش‌های صورت گرفته در زمینه تشخیص برنامک‌های بازبسته‌بندی در این فصل، می‌توان گفت که اکثر پژوهش‌های موجود در این زمینه، نیازمند تقویت ایده‌ی اولیه هستند و هنوز پژوهش کاملی در این زمینه که علاوه بر دقت بالا، سرعت تشخیص مناسبی نیز داشته باشد، توسعه داده نشده‌است. اما به صورت کلی، در مقایسه‌ی روش‌های پویا و ایستا می‌توان گفت که عمده‌ی روش‌های موجود با تمرکز بر **تحلیل ایستا** سعی در افزایش کارایی روش‌های پیشین دارند. علت این موضوع آن است که تحلیل ایستا پاسخ‌گوی طیف وسیعی از برنامک‌های اندرویدی خصوصا برنامک‌های رایگان است در حالی که پژوهش‌های موجود در زمینه‌ی تحلیل پویا، اکثرا از مجموعه‌ی داده‌ی آزمون کوچک و محدودی، خصوصا در زمینه‌ی برنامک‌های تجاری، استفاده کرده‌اند. علت دیگر این موضوع آن است که تهدید اصلی کاربران در زمینه‌ی برنامک‌های اندرویدی بازبسته‌بندی شده، از سمت برنامک‌های رایگان رخ می‌دهد و برنامک‌های غیررایگان و تجاری، معمولا پیش از توسعه به طور کاملی از روش‌های جلوگیری از بازبسته‌بندی استفاده می‌کنند. پژوهش پیاده‌سازی شده در این پایان‌نامه، مبتنی بر تحلیل ایستا با استفاده از گراف و استخراج ویژگی‌های مقاوم در مقابل مبهم‌نگاری است، بنابراین در ادامه‌ی این پژوهش، مقایسه‌ی روش‌های مبتنی بر تحلیل ایستا را انجام خواهیم داد. اگر به طور کلی دو معیار دقت و سرعت اجرا را در دسته روش‌های مبتنی بر تحلیل ایستا در نظر بگیریم، روش‌های مبتنی بر گراف از دقت بالایی در تشخیص برخوردار هستند چرا که تغییر گراف‌های حاصل از فراخوانی و یا اجرای‌ی برنامک سخت و هزینه‌بر است اما به صورت کلی، از آن‌جایی که الگوریتم‌های مقایسه‌ی گرافی، به خصوص الگوریتم‌های تشخیص گراف هم‌ریخت، سرعت بسیار پایینی در مقایسه دارند، معمولا روش‌های این دسته به قدری کند هستند که عملا استفاده از آن‌ها در یک ایزار کاربردی و محیط صنعتی ناممکن خواهد بود. از طرفی روش‌های مبتنی بر تحلیل ترافیک و تحلیل منابع، خصوصا در روش‌هایی که از طبقه‌بندی استفاده می‌شود، از سرعت بالایی

در تشخیص برخوردار هستند اما به جهت این‌که ایجاد مبهم‌نگاری در منابع برنامک آسان و کم‌هزینه‌است، بنابراین بازبسته‌بندی ممکن است مبتنی بر منابع برنامک نیز اتفاق بیافتد. علاوه بر این، هدف بسیاری از برنامک‌های بازبسته‌بندی، فریب کاربران ناآگاه با استفاده از ایجاد یک واسط کاربری مشابه با برنامک اصلی است، بنابراین بررسی این دسته از ویژگی‌ها می‌تواند ما را در تشخیص برنامک‌های اندرویدی بازبسته‌بندی شده یاری دهد. در جدول ۳ـ۲ پژوهش‌های مطرح مبتنی بر تحلیل ایستا را از نظر سه خاصیت یعنی **جست‌وجوی دودویی، سربار زمانی و دقت تشخیص** مورد مقایسه قرار داده‌ایم.

جدول ۳ـ۲: مقایسه‌ی روش‌های مبتنی بر تحلیل ایستا

دسته‌بندی روش‌ها	پژوهش‌کنندگان	سربار محاسباتی	دقت تشخیص	مقایسه‌ی دودویی
مبتنی بر آپکد و دنباله‌ی دستورات	ژو وهمکاران[۳۷]	متوسط	پایین	✓
	ژو و همکاران[۲۸]	متوسط	پایین	✓
	دزنور و همکاران[۳۸]	متوسط	پایین	✓
	جرومه و همکاران[۴۱]	متوسط	پایین	×
	سرنیل و همکاران[۴۲]	متوسط	پایین	×
	لین و همکاران[۴۳]	متوسط	پایین	×
مبتنی بر گراف	سان و همکاران[۱۰]	زیاد	خوب	✓
	هو و همکاران[۴۹]	متوسط	خوب	✓
	ژو و همکاران[۵۰]	زیاد	خوب	✓
	چن و همکاران[۵۱]	کم	خوب	✓
	جنگ و همکاران[۵۲]	زیاد	خوب	✓
	وانگ[۵۳]	زیاد	خوب	✓
	ترکی[۵]	زیاد	خوب	✓
مبتنی بر تحلیل ترافیک شبکه	الشهری و همکاران[۵۵]	متوسط	پایین	✓
	هه و همکاران[۵۶]	متوسط	پایین	×
	مالک و همکاران[۵۸]	متوسط	پایین	✓
	ایلند و همکاران[۵۹]	متوسط	پایین	✓
	شارما و همکاران[۶۰]	متوسط	متوسط	×
مبتنی بر تحلیل منابع	شائو و همکاران[۱]	کم	خوب	×
	سان و همکاران[۶۲]	زیاد	خوب	✓
	هو و همکاران[۶۳]	متوسط	خوب	✓
	لین و همکاران[۶۴]	متوسط	پایین	✓
	گوآ و همکاران[۶۵]	زیاد	متوسط	✓
	لیو و همکاران[۶۶]	متوسط	خوب	✓
	ما و همکاران[۶۹]	متوسط	خوب	✓

فصل ۴

راهکار پیشنهادی

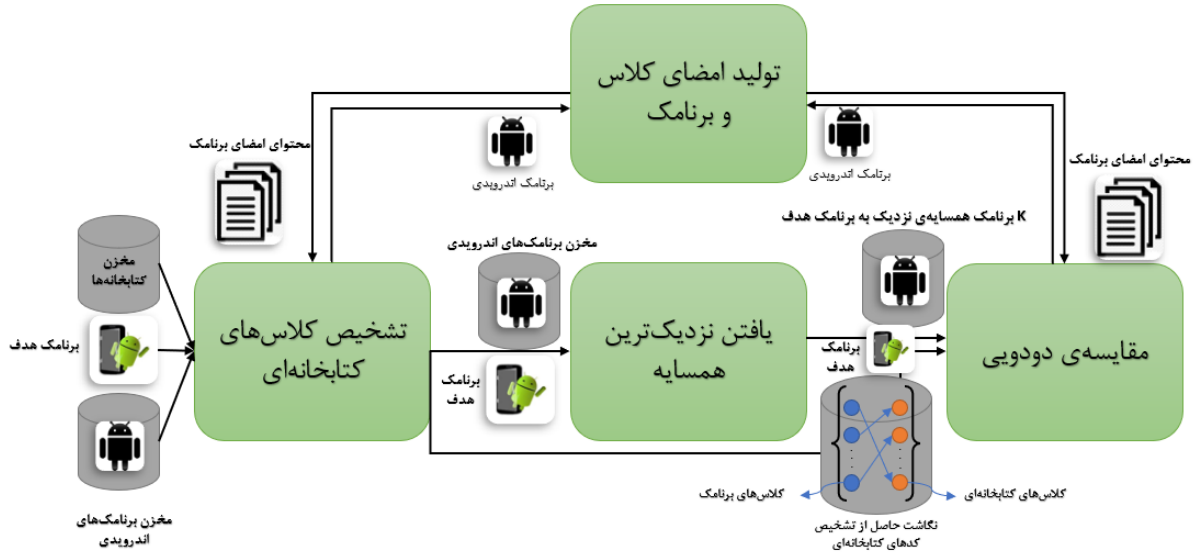
تقریباً در تمامی راهکارهای ارائه‌شده جهت تشخیص برنامک‌های بازبسته‌بندی شده، نیازمند انجام مقایسه‌ی دودویی ویژگی‌های استخراج‌شده از برنامک‌ها در راستای تشخیص جفت تقلبی[۱] هستیم. به همین جهت اصولا تشخیص جفت برنامک بازبسته‌بندی شده از میان مخزن برنامک‌ها روشی زمان‌بر و پرهزینه‌است. از طرفی همانطور که بررسی شد، برای تشخیص برنامک بازبسته‌بندی شده، نیازمند ویژگی‌هایی به صورت ایستا و یا پویا از برنامک‌ها هستیم که رفتار منطقی آن‌ها را نشان دهد. از بررسی کارهای پیشین و مطالعه در زمینه‌ی تحلیل برنامک‌های اندرویدی، می‌توان دریافت که پارامترهای دقت و سرعت در تشخیص برنامک‌های بازبسته‌بندی شده، رابطه‌ی مستقیمی با ویژگی‌های منتخب به جهت مقایسه‌ی دودویی برنامک‌ها دارد. پیچیدگی ویژگی‌های منتخب منجر به کاهش سرعت و مقاومت آن‌ها در مقابل روش‌های مبهم‌نگاری، در نهایت منجر به افزایش دقت در تشخیص جفت تقلبی خواهد شد. در این فصل ابتدا نمای کلی از پژوهش را توضیح می‌دهیم و مولفه‌های اصلی پژوهش را به صورت مختصر بررسی خواهیم کرد. سپس هر کدام از مولفه‌های مطروحه را با جزئیات بیشتری بررسی و معایب و مزایای هر کدام را شرح خواهیم داد.

۴ـ۱ راهکار پیشنهادی

از آن‌جایی که مقایسه‌ی دودویی برنامک ورودی با تمامی برنامک‌های موجود در مخزن، هزینه‌ی محاسباتی زیادی را در روند تشخیص تحمیل می‌کند، استفاده از یک مرحله پیش‌پردازش[۲] می‌تواند به افزایش سرعت تشخیص کمک کند. همانطور که در شکل ۴ـ۱ مشخص شده‌است، راهکار پیشنهادی در این پژوهش به صورت کلی از سه مؤلفه‌ی «تشخیص کتابخانه‌های اندرویدی» ، «یافتن نزدیک‌ترین همسایه» و «تشخیص

[۱] Repackaged Pair
[۲] Preprocess

برنامک بازبسته‌بندی شده» تشکیل شده‌است. مؤلفه‌ی «تشخیص کتابخانه‌های اندرویدی» با الهام از پژوهش [۵] و همراه با تغییر امضای کلاسی به جهت افزایش سرعت، پیاده‌سازی شده‌است. مخزن کتاب‌خانه‌های اندرویدی و برنامک هدف، به عنوان ورودی به این مؤلفه داده می‌شوند. به صورت کلی در این مؤلفه کلاس‌های هربرنامک، با تمامی کلاس‌های موجود در مخزن کتابخانه‌ها مقایسه می‌شود. برای کاهش فضای مقایسه‌ی کتابخانه‌ها با کلاس‌های برنامک هدف، از «ماژول فیلتر کتابخانه‌ها» استفاده شده‌است. این ماژول تعداد کلاس‌های مورد بررسی در هر کتابخانه را با استفاده از دو «فیلتر ساختاری» و «طول امضا» کاهش می‌دهد و در نهایت با استفاده از توابع چکیده‌سازی محلی[۳]، مقایسه‌ی کلاس‌های برنامک هدف و مخزن کتابخانه‌های اندرویدی در ماژول «استخراج نگاشت» انجام می‌شود. حل مسئله‌ی نگاشت میان کلاس‌های برنامک و کلاس‌های کتابخانه‌ای، همان حل مسئله‌ی تخصیص[۴] است که در آن نگاشتی از کمترین هزینه (مبتنی بر چکیده‌سازی محلی) از یک گراف دوبخشی استخراج می‌شود.



شکل ۴ـ۱: نمای کلی راهکار پیشنهادی

در این پژوهش از یک طبقه‌بند نزدیک‌ترین همسایه[۵] به عنوان مرحله‌ی پیش‌پردازش ورودی‌ها استفاده شده‌است. در این مؤلفه با استفاده از تعدادی ویژگی مبتنی بر واسط‌های کاربری[۶]، برنامک‌های موجود در مخزن و برنامک ورودی هدف طبقه‌بندی[۷] و تعدادی از نزدیک‌ترین همسایه‌های برنامک ورودی به عنوان برنامک‌های مشکوک به جهت بررسی دقیق‌تر به مرحله‌ی بعد خواهند رفت.

---

[۳] Fuzzy Hashing
[۴] Assignment Problem
[۵] Nearest neighbor
[۶] User Interface
[۷] Clustering

مؤلفه‌ی دیگری که در این پژوهش پیاده‌سازی شده است، تشخیص برنامک‌های بازبسته‌بندی شده است. این مؤلفه، در یک مرحله پیش‌پردازش با استفاده از مؤلفه‌ی تشخیص کتابخانه‌ها، نگاشتی از کلاس‌های برنامک و کلاس‌های کتابخانه‌ای مخزن به دست می‌آورد. سپس دو برنامک مورد بررسی و نگاشتی از کلاس‌های هردو، به عنوان ورودی ماژول طبقه‌بند، وارد می‌شود. در مؤلفه‌ی تشخیص برنامک‌های بازبسته‌بندی شده، ابتدا کدهای کتابخانه‌ای با استفاده از نگاشت حاصل از مرحله‌ی قبلی حذف می‌شوند و کلاس‌های باقی‌مانده برای ساخت امضا توسط ماژول ساخت امضای برنامک، تحلیل می‌شوند. در نهایت با استفاده از چکیده‌سازی محلی و تعیین یک حدآستانه، تعیین بازبسته‌بندی صورت می‌گیرد.

۴ـ۲ ساخت امضای برنامک

هدف از ساخت امضای[۸] برنامک، تشکیل مدلی از رفتار منطقی و دستوری آن می‌باشد به طوری که مقاومت بالایی در مقابل راهکارهای مبهم‌نگاری داشته‌باشد. برای ساخت امضای برنامک از الحاق امضای تمامی کلاس‌های آن استفاده می‌شود. در این پژوهش ، مؤلفه‌های تشخیص کتابخانه‌های اندرویدی و تشخیص بازبسته‌بندی، هر دو از امضای کلاس برای شباهت‌سنجی استفاده کرده‌اند. امضای کلاس متشکل از مهم‌ترین ویژگی‌های آن است که علاوه بر مدل‌کردن رفتار کلاس، مقاومت بالایی در مقابل روش‌های مبهم‌نگاری داشته‌باشد. برای تشخیص کدهای کتابخانه‌ای، امضای تمامی کلاس‌های جاوا در کتابخانه‌ی مذکور ساخته‌شده و برای تشخیص قسمت‌هایی که شامل کدهای کتابخانه‌ای در برنامک اندرویدی است استفاده می‌شود.

از آن‌جایی که امضا برای هر کلاس تولید می‌شود، بنابراین به صورت کلی دو مرحله برای تولید امضای کلاس پیاده‌سازی شده‌است. در ابتدا با استفاده از ویژگی‌های مبتنی بر متد[۹]، امضای متدهای پویای[۱۰] کلاس استحراج شده و سپس با استفاده از طرحی که در ادامه توضیح داده خواهدشد امضای کلاس ساخته می‌شود.در ادامه ابتدا توصیف کلی از تمامی قسمت‌های امضای کلاسی در برنامک ارائه می‌دهیم و دلیل استفاده از این پارامتر را در امضای کلاس عنوان می‌کنیم. سپس ساختار صوری[۱۱] تشکیل امضای کلاس‌های برنامک و در نهایت امضای برنامک را شرح می‌دهیم.

---

[۸] Signature
[۹] Method
[۱۰] Dynamic Method
[۱۱] Formal

۴ـ۲ـ۱ توصیف پارامترهای امضای برنامک

در این بخش ویژگی‌های استفاده‌شده در امضای کلاس را شرح می‌دهیم و دلایل استفاده از آن‌ها را عنوان می‌کنیم.

- **تغییردهنده‌ی متد:** تغییردهنده‌های هر متد[۱۲]، مشخص‌کننده‌ی چگونگی پیاده‌سازی متدهای جاوا هستند که اکثرا به سختی قابل تغییر می‌باشند چرا که در صورت تغییر آن‌ها بدنه‌ی متد[۱۳] معمولاً باید به صورت کلی تغییر کند، بنابراین استفاده از این ویژگی، مقاومت بالایی مقابل مبهم‌نگاری را به ارمغان می‌آورد. برای ساخت امضای متد از چهار نوع تغییردهنده‌ی بومی[۱۴]، انتزاعی[۱۵]، ایستا[۱۶] و سازنده[۱۷] استفاده شده‌است. متدهای بومی، شامل دستورات به زبان بومی جاوا هستند که توسط پردازنده به صورت مستقیم اجرا می‌شود. متدهای ایستا توسط تمامی نمونه‌های کلاس قابل دسترسی هستند و تنها به متغیرهای ایستا در کلاس دسترسی دارند. متدهای سازنده، در واقع وظیفه‌ی مقداردهی اولیه به متغیرهای پویا در هنگام ساخت نمونه‌ی کلاسی[۱۸] در برنامک‌های اندرویدی را بر عهده دارند.

- **نوع داده‌ی خروجی و ورودی متد:** ویژگی دیگری که در امضای متد مورد استفاده قرار گرفته‌است، نوع داده‌ی متغیرهای خروجی و ورودی به هر متد می‌باشد. به دلیل آنکه تغییر ورودی و خروجی متدها غالبا بدون تغییر بدنه‌ی اصلی توابع امکان‌پذیر نیست، بنابراین تغییر آن‌ها منجر به صرف هزینه‌ی زیادی برای متقلبان می‌شود، چرا که در این صورت نیاز به تغییر بدنه‌ی متد دارند، به طوری که منطق اصلی متد حفظ شود.

- **متدهای فراخوانی‌شده:** در این قسمت از دو دسته‌ی مهم از متدهایی که در بدنه‌ی برنامک فراخوانی‌شده‌اند استفاده شده‌است. دسته‌ی اول متدهایی هستند که متدهای کتابخانه‌ای جاوا نیستند و به صورت غیرایستا، توسط توسعه‌دهنده پیاده‌سازی شده‌اند. لازم به ذکر است که به دلیل آنکه در قسمت تشخیص برنامک‌های اندرویدی بازبسته‌بندی شده، کدهای کتابخانه‌ای حذف شده‌اند، بنابراین تعریف متدهای فراخوانی شده در این قسمت، بدون در نظر گرفتن متدهای کتابخانه‌ای اندرویدی خواهد بود. دسته‌ی دوم، متدهای زبان جاوا هستند که داخل بسته‌ی کتابخانه‌ای جاوا حضور دارند. تغییر این دسته از متدها آسان‌تر از متدهای قسمت قبلی است، اما در هر حال نمی‌توان بدون پرداخت هزینه‌ی محاسباتی آن‌ها را حذف کرد.

---

[۱۲] Method Modifier
[۱۳] Method Body
[۱۴] Native
[۱۵] Abstract
[۱۶] Static
[۱۷] Constructor
[۱۸] Class Instance

- **فراخوانی واسط‌های برنامه‌نویسی:** واسط‌های برنامه‌نویسی[۱۹]، هسته‌ی اصلی رفتار هر متد در برنامک‌های اندرویدی هستند که بدافزارنویسان نیز از آن‌ها استفاده می‌کنند. تغییر این دسته از فراخوانی‌ها در متدهای کلاسی، سخت و با پیچیدگی همراه است.

در امضای هر کلاس نیز از ویژگی‌های زیر استفاده شده‌است:

- **هسته‌ی کلاس:** هسته‌ی کلاس شامل امضای تمامی متدهای موجود با استفاده از ویژگی‌های بخش قبل می‌باشد.
- **کلاس‌های داخلی:** از آن‌جا که تغییر سلسله مراتب کلاس‌های جاوایی، اکثرا ناممکن است بنابراین امضای تمامی کلاس‌های داخلی به همراه متدهای پیاده‌سازی شده در این قسمت استفاده می‌شود. تغییر سلسله مراتب کلاس‌های داخلی نیازمند تغییر تمامی فراخوانی‌های موجود از کلاس‌های مورد نظر است بنابراین تغییر سلسله مراتب تقریبا ممکن نیست.
- **سطح کلاس:** همانطور که گفته‌شد، تغییر سلسله مراتب کلاس‌ها سخت و زمان‌بر است بنابراین شماره‌ی سطح کلاس در ساختار سلسله‌مراتبی، ویژگی مقاوم دیگری است که در این قسمت استفاده می‌شود.
- **کلاس پدر:** تغییر ساختار ارث‌بری کلاس‌ها[۲۰]، هزینه‌بر است چرا که تغییر این ساختار نیازمند تغییر فراخوانی‌های متعدد ناشی از ساختن نمونه‌های کلاسی است. بنابراین در امضای هر کلاس، از امضای پدر آن کلاس نیز استفاده شده‌است.
- **نام کلاس داخلی و بیرونی:** برای بررسی ریزدانه‌تر[۲۱]، علاوه بر سطح کلاس، از نام کلاس‌های درونی و بیرونی نیز استفاده شده‌است. اکثر مبهم‌نگارها، از بذر یکسانی برای تولید نام‌های تصادفی استفاده می‌کنند بنابراین ایجاد نام‌های متنوع از یک نام نیازمند تغییر نام اولیه‌ی متد است.
- **طول هسته‌ی کلاس:** طول هسته‌ی کلاس، شامل متدهای کلاسی، ویژگی دیگری است که در امضای کلاس مورد استفاده قرار گرفته‌است.

---

[۱۹] Application Programming Interface
[۲۰] Class inheritance
[۲۱] Fine Grain

۴ـ۲ـ۲ توصیف صوری امضای متد

به صورت کلی امضای متد از ۵ قسمت متفاوت تشکیل شده‌است:

$$\langle Modifier, RetType, InputType, JLibMethodCallee, \\ NonStaticAppMethodCallee, ApiCallSootSignature\rangle \quad (۱ـ۴)$$

- $Modifier$: در این بخش رشته‌ای شامل تغییردهنده قرار می‌گیرد. همانطور که در بخش پیشین بررسی کردیم، تغییردهنده‌ی متد، شامل یکی از انواع بومی، انتزاعی، ایستا و یا سازنده‌است. در صورتی که تغییردهنده‌ی متد هیچ‌کدام از موارد ذکر شده نباشد، مقدار $Null$ به جای این ویژگی قرار می‌گیرد.

- $RetType$: نوع داده‌ی بازگشتی از متد، مرتب‌شده به صورت الفبایی، به صورت رشته در این قسمت قرار می‌گیرد. در صورتی که نوع داده‌ی متد، از انواع داده‌ای زبان جاوا نباشد، نام کلاس این نوع داده‌ای به صورت رشته جایگزین می‌شود.

- $InputType$: نوع داده ورودی‌های متد، مرتب شده به صورت الفبایی، به صورت رشته در این قسمت قرار می‌گیرند. در صورت نبودن نوع داده در زبان جاوا، همانند $RetType$ عمل می‌شود.

- $JLibMethodCallee$: رشته‌ای متشکل از تمامی توابع کتابخانه‌ای جاوا، که در بدنه‌ی اصلی متد صدا زده‌شده‌اند در این قسمت قرار می‌گیرد. رشته‌ی حاصل به صورت الفبایی مرتب شده‌است.

- $NonStaticAppMethodCallee$: رشته‌ی مرتب شده به صورت الفبایی شامل نام تمامی توابع غیر کتابخانه‌ای و غیرایستا که توسط تابع فراخوانی شده‌اند. لازم به ذکر است، از آن‌جایی که فراخوانی توابع ایستا، مقاومت بالایی در مقابل مبهم‌نگاری ندارد، در این قسمت از توابع غیر ایستا استفاده شده‌است. افزودن فراخوانی توابع ایستا، یکی از روش‌های محبوب در مبهم‌نگاری برنامک‌های اندرویدی است.

- $ApiCallSootSignature$: در این قسمت رشته‌ای از سه ویژگی مهم توابع واسط‌های برنامه‌نویسی، شامل نوع کلاس متد فراخوانی، کلاس نوع بازگشتی متد و نام متد قرار می‌گیرد. تغییر نام واسط‌های برنامه‌نویسی قطعا نیازمند تغییر کتابخانه‌های اندرویدی و ایجاد مبهم‌نگاری در آن‌ها است، به همین دلیل در این قسمت نام متد، تا حدودی مقاوم در مقابل مبهم‌نگاری برنامک می‌باشد.

$$\langle DeclaringClass, RetType, MethodName\rangle \quad (۲ـ۴)$$

۴ـ۲ـ۳ توصیف صوری امضای کلاس

امضای کلاس، شامل ۶ قسمت است که در ادامه به بررسی هر کدام از آن‌ها می‌پردازیم:

$$\langle ClassCoreSig, InnerClassesSig, InheritedClassesSig, ImpInterfacesSig \\ , ClassLevel, InnerOuterClassName, ClassLen, NumofInnerClass\rangle \quad (۴ـ۳)$$

- $ClassCoreSig$: رشته‌ی حاصل از الحاق امضای تمامی متدهای کلاس در این قسمت قرار می‌گیرد. ترتیب قرارگرفتن امضای متدهای کلاس، بر اساس خطوطی است که پیاده‌سازی شده‌اند، یعنی اگر متد A در کلاس B پیش از متد C پیاده‌سازی شده‌بود، آنگاه امضای متد A پیش از متد C قرار خواهد گرفت.

$$\langle MethodSig_۱, MethodSig_۲, ..., MethodSig_n\rangle \quad (۴ـ۴)$$

- $InnerClassesSig$: رشته‌ی حاصل از الحاق امضای کلاس‌های داخلی کلاس مورد نظر، مرتب‌شده به صورت الفبایی در این قسمت قرار خواهد گرفت.

- $InheritedClassesSig$: در صورتی که کلاس پدر[۲۲]، یکی از کلاس‌های کتابخانه‌ای جاوا باشد، نام کلاس پدر در این قسمت قرار می‌گیرد، اما در غیر این صورت، امضای کلاس پدر، شامل $CoreClassesSig$ و $InnerClassesSig$ به صورت رشته در این قسمت قرار می‌گیرد.

- $ImpInterfacesSig$: در صورتی که واسط پیاده‌سازی شده[۲۳]، از نوع کلاس‌های کتابخانه‌ای جاوا باشد، نام آن به صورت رشته در این قسمت قرار خواهد گرفت و در غیر این صورت همانند قسمت قبل، امضای واسط‌ها، شامل الحاق $CoreClassesSig$ و $InnerClassesSig$ تمامی آن‌ها، مرتب شده به صورت الفبایی خواهد بود.

- $ClassLevel$: عمق کلاس[۲۴] در ساختار سلسله‌مراتبی کلاس‌های هر بسته به صورت عددی در این قسمت قرار خواهد گرفت. برای مثال اگر کلاس مورد نظر، ریشه‌ی درخت سلسله‌مراتبی بسته‌ها باشد، عدد ۰ و اگر فقط یک کلاس بیرونی داشته باشد ، عدد ۱ در این قسمت قرار خواهد گرفت.

- $InnerOuterClassesName$: نام کلاس‌بیرونی و کلاس‌های درونی مرتب‌شده به صورت الفبایی به صورت رشته در این قسمت قرار خواهد گرفت.

---

[۲۲] Parent Class
[۲۳] Implemented Interface
[۲۴] Class's Depth

- $ClassLen$: در این قسمت طول رشته‌ی امضای کلاس، به صورت عددی درج می‌شود.
- $NumOfInnerClass$: تعداد کلاس‌های داخلی کلاس هدف، به صورت عددی در این قسمت قرار می‌گیرد.

۴ـ۲ـ۴ توصیف صوری امضای برنامک

پس از تشکیل امضای تمامی کلاس‌های برنامک با استفاده از امضای هر متد، امضای برنامک با استفاده از الحاق امضای تمامی کلاس‌های آن ساخته می‌شود.

$$\langle ClassSig_1, ClassSig_2, ..., Classsig_n \rangle \quad (۵ـ۴)$$

۴ـ۳ مؤلفه‌های تشخیص برنامک‌های بازبسته‌بندی شده

در این قسمت به بررسی ماژول[۲۵]های هر کدام از سه مؤلفه‌ی تشخیص برنامک‌های بازبسته‌بندی شده می‌پردازیم.

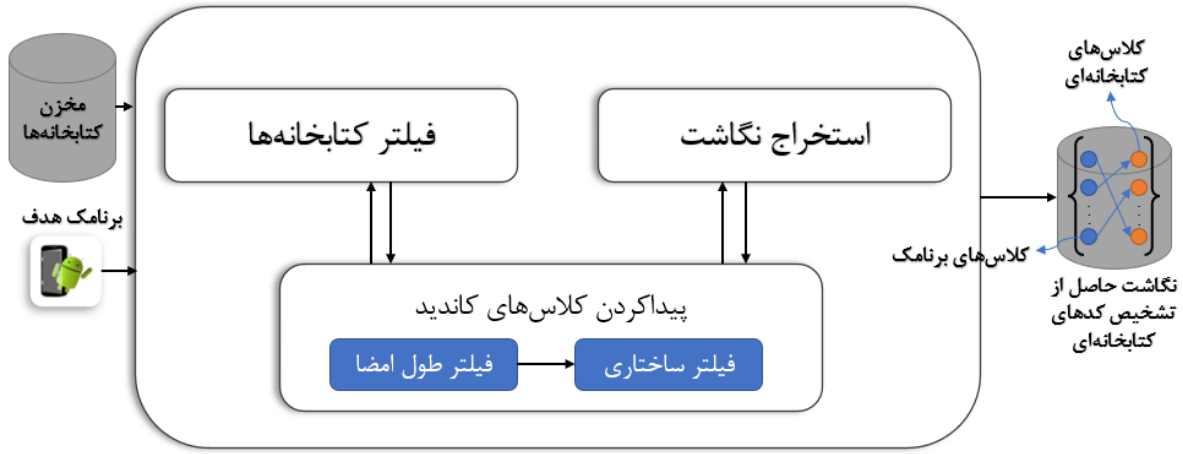
۴ـ۳ـ۱ مؤلفه‌ی تشخیص کدهای کتابخانه‌ای

با توجه به تصویر ۴ـ۲، این مؤلفه دارای سه ماژول اصلی یافتن کلاس‌های کاندید[۲۶]، فیلتر کلاس‌های کاندید مبتنی بر فیلترهای ساختاری[۲۷] و طول امضا و در نهایت ماژول یافتن نگاشت[۲۸] میان کلاس‌های برنامک و کلاس‌های کتابخانه‌ای است.

آسان ترین روش یافتن کلاس‌های کتابخانه‌ای مقایسه‌ی دودویی کلاس‌های موجود در مخزن کتابخانه‌ها با تمامی کلاس‌های برنامک می‌باشد اما این روش بسیار پرهزینه است، به همین جهت در مرحله‌ی اول از تشخیص کدهای کتابخانه‌ای با استفاده از دو فیلتر ساختاری و طولی، تعداد کلاس‌های مورد مقایسه با برنامک مقصد را کاهش می‌دهیم. در قسمت ابتدایی توسط فیلتر طولی، تنها کلاس‌هایی را بررسی خواهیم کرد که طول آن‌ها از کلاس مبدا از یک حد آستانه[۲۹] Thr_L فاصله داشته‌باشد. در قسمت فیلتر ساختاری، با استفاده از بررسی و تحلیل چندین ویژگی روی کلاس‌های کتابخانه‌ای، مجموعه‌ی کلاس‌هایی که در

[۲۵] Module
[۲۶] Candidate
[۲۷] Structural
[۲۸] Mapping
[۲۹] Threshold

خروجی فیلتر طولی ظاهر شده‌اند را مجدداً کاهش می‌دهیم. ویژگی‌های مورد بررسی در این قسمت شامل تعداد کلاس‌های بیرونی و داخلی، ارث‌بری کلاسی کلاس هدف، ارث‌بری از کلاس‌های کتابخانه‌ای جاوا و یا اندروید، تعداد واسط‌های پیاده‌سازی شده و معادل بودن کلاس‌های بیرونی می‌باشد. کلاس خروجی پس از فیلتر طولی، تنها در صورتی بعد از اعمال فیلتر ساختاری نیز حضور خواهد داشت که در تمامی ویژگی‌های ذکر شده با کلاس هدف تطابق داشته‌باشد. شبه‌کد این رویه در الگوریتم ۱ مشخص شده‌است.



شکل ۴ـ۲: نمای کلی مؤلفه‌ی تشخیص کدهای کتابخانه‌ای

در ماژول پیداکردن کلاس‌های کاندید، تمامی کلاس‌های برنامک و یک کلاس هدف از میان کلاس‌های کتابخانه‌ای به عنوان ورودی به ماژول مورد نظر ارسال می‌شود و در نهایت لیستی از کلاس‌های برنامک به عنوان کلاس‌های مشابه در خروجی مشخص می‌شود. شبه‌کد رویه‌ی پیداکردن کلاس‌های کاندید در الگوریتم ۲ مشخص شده‌است. پس از اجرای ماژول پیداکردن کلاس‌های کاندید، کتابخانه‌هایی که که کلاسی مشابه در برنامک دارند را شناسایی می‌کنیم. به جهت شناسایی کتابخانه‌های موجود در برنامک، پیش از اجرای مولفه‌ی تشخیص کدهای کتابخانه‌ای، مخزنی شامل کتابخانه‌های اندرویدی مشهور ایجاد می‌کنیم و تمامی کلاس‌های کتابخانه‌ای را در این مخزن قرار می‌دهیم. سپس تعداد N کلاس با طول بیشینه از هر کتابخانه را انتخاب و به عنوان کلاس‌های هدف قرار می‌دهیم. در ادامه، الگوریتم پیداکردن کلاس‌های کاندید را برای هر کدام از کلاس‌های هدف کتابخانه‌ی lib اجرا کرده و تعدادی کلاس کاندید از میان کلاس‌های برنامک انتخاب می‌کنیم. در این مرحله، کلاس‌های کاندید برنامک را با کلاس $Class$ از کتابخانه‌ی lib با استفاده از روش‌های چکیده‌سازی محلی مقایسه کرده و در صورتی که میزان شباهت دو کلاس بیش از یک حد آستانه Thr_f باشد، کلاس کاندید مورد نظر با کلاس $Class$ مطابقت یافته و

الگوریتم ۱ فیلتر کتابخانه‌ها

ورودی: کلاس هدف برای یافتن لیستی از کلاس‌های مشابه با آن t_c، لیستیی از کلاس‌های برنامک S_{app}

خروجی: لیستی از کلاس‌های کاندید برنامک مشابه با کلاس هدف t_c

۱: قرار بده $Candidate = \emptyset$

۲: قرار بده $L = length(Sig_{t_c})$ ◁ *مقدار $Sig(t_c)$ امضای نهایی کلاس t_c می‌باشد*

۳: ◁ ***فیلتر طولی کلاس‌های برنامک***

۴: به ازای $Class \in S_{app}$:

۵: اگر $L - T_L < length(Sig_{Class}) < L + T_L$:

۶: $Candidate \cup Class$

۷: ◁ ***فیلتر ساختاری کلاس‌های برنامک***

۸: به ازای $Class \in Candidate$:

۹: قرار بده $Condition = True$

۱۰: قرار بده $Condition = Condition \wedge (\#outerClass(Class) = \#outerClass(t_c))$

۱۱: قرار بده $Condition = Condition \wedge (\#InnerClass(Class) = \#InnerClass(t_c))$

۱۲: قرار بده $Condition = Condition \wedge (DoesInherit(Class) = DoesInherit(t_c))$

۱۳: قرار بده $Condition = Condition \wedge (SDKInherit(Class) = SDKInherit(t_c))$

۱۴: قرار بده $Condition = Condition \wedge (\#Interfaces(Class) = \#Interfaces(t_c))$

۱۵: قرار بده $Condition = Condition \wedge (SDKInterface(Class) = SDKInterface(t_c))$

۱۶: قرار بده $Condition = Condition \wedge (OuterClass(Class) = OuterClass(t_c))$

۱۷: اگر $Condition \neq True$:

۱۸: $Condidate = Condidate - \{Class\}$

برای نگاشت میان کلاس‌های کتابخانه‌ای و کلاس‌های برنامک، به مرحله‌ی بعد خواهد رفت.

الگوریتم ۲ پیداکردن کلاس‌های کاندید

ورودی: لیست کلاس‌های برنامک S_{app}، مخزن کتابخانه‌های $libs$

خروجی: لیست کتابخانه‌های استفاده‌شده در برنامک

۱: قرار بده $Matched = \emptyset, Libs = \emptyset$

۲: به ازای $lib \in libs$:

۳: قرار بده $GetMaxLengthClasses(lib, N) = TargetClasses$

۴: به ازای $Class \in TargetClasses$:

۵: قرار بده $Candidates = FindCandidate(Class, S_{app}), Matched_{Class} = \emptyset$

۶: به ازای $Ca \in Candidates$:

۷: اگر $FHashCompare(Sig_{ca}, Sig(Class)) > Thr_F$:

۸: قرار بده $Matched_{Class} = Matched_{Class} \cup Ca$

۹: قرار بده $Matched = Matched \cup Matched_{class}$

۱۰: اگر $Matched \neq \emptyset$:

۱۱: قرار بده $Libs = Libs \cup lib$

۱۲: برگردان $Libs$

ماژول استخراج نگاشت، به جهت ایجاد نگاشتی میان کلاس‌های کتابخانه‌ای (خروجی ماژول پیداکردن کلاس‌های کاندید) و کلاس‌های برنامک پیاده‌سازی شده‌است. ورودی ماژول لیست Cls_{lib} شامل کلاس‌های کتابخانه‌ای برنامک، حاصل از اجرای الگوریتم ۲ و لیست Cls_{app} کلاس‌های برنامک هدف می‌باشد. خروجی ماژول، نگاشتی یک به یک و پوشا از کلاس‌های کتابخانه‌ای به کلاس‌های برنامک می‌باشد و مشخص می‌کند کدام یک از کلاس‌های برنامک، کلاس‌های کتابخانه‌ای هستند. به عبارت دیگر، خروجی ماژول استخراج نگاشت، نگاشتی به صورت زیر می‌باشد:

$$f : L \rightarrow A \qquad L \subseteq Cls_{lib} \quad , A \subseteq Cls_{app} \tag{۶ـ۴}$$

در ابتدای این ماژول، کلاس‌های کتابخانه‌ای در سطح i ام را با استفاده از متد $FilterClassByLevel$ محاسبه می‌کنیم و آن را با $S_{lib,i}$ نمایش می‌دهیم. در ادامه به ازای تمامی کلاس‌های موجود در مجموعه‌ی کلاسی $S_{lib,i}$، کلاس‌های کاندید را با استفاده از ماژول یافتن کلاس‌های کاندید محاسبه و تحت عنوان

$Candidate_{cls}$ ذخیره می‌کنیم. سپس برای محاسبه‌ی کلاس‌های منطبق[۳۰] بر کلاس هدف، تمامی کلاس‌های کاندید حاصل از مرحله‌ی قبل را با استفاده از توابع چکیده‌سازی محلی، با کلاس هدف مقایسه و در صورتی که میزان تشابه دو کلاس از حد آستانه‌ی Thr_s بیشتر بود آنگاه کلاس مورد نظر را به عنوان کلاس تطابق‌داده شده به مجموعه‌ی $Matched_{Class}$ اضافه می‌کنیم.

در ادامه سعی می‌شود تا مسأله‌ی پیداکردن کلاس‌های کتابخانه‌ای را به مسأله‌ی تخصیص در یک گراف دوبخشی (حاصل از کلاس‌های کتابخانه‌ای و کلاس‌های برنامک) تبدیل کرد. به همین جهت گراف دو بخشی وزن‌دار G با تابع هزینه‌ی $C \times V \rightarrow \mathbb{N}$ را با استفاده از کلاس‌های کتابخانه‌ای و کلاس‌های برنامک تشکیل می‌دهیم. گراف G به صورت زیر تعریف می‌شود:

$$G = (U, V, E) \qquad (۴\text{ـ}۷)$$

مجموعه‌ی گره‌های U شامل امضای کلاس‌های هدف کتابخانه‌ای در سطح i ام است و گره‌های V شامل تمامی کلاس‌های تطابق‌یافته با کلاس هدف cls می‌باشد. مجموعه یال‌های گراف E، حاصل از ایجاد تطابق میان کلاس‌های برنامک و کلاس‌های کتابخانه‌ای و $FhashCompare$ متد مقایسه‌ی امضای کلاسی با استفاده از چکیده‌سازی محلی می‌باشد. متد مقایسه به ازای ورودی امضای کلاس‌های مورد مقایسه، عددی بین ۰ و ۱۰۰ را بازمی‌گرداند.

$$U = \{cls \,|cls \in S_{lib,i} \wedge Matched_{cls} \neq \emptyset\} \qquad (۴\text{ـ}۸)$$

$$V = \bigcup_{cls \in U} Matched_{cls} \qquad (۴\text{ـ}۹)$$

$$E = \{(cls, c) \,|cls \in \ s_{lib,i} \ , c \in Matched_{cls}\} \qquad (۴\text{ـ}۱۰)$$

$$C(cls, c) = ۱۰۰ - FhashCompare(Sig_c, Sig_{cls}) \qquad (۴\text{ـ}۱۱)$$

$$FhashCompare : Sig_۱ \times Sig_۲ \rightarrow R \quad Sig_۱, Sig_۲ \in ClassesSigs, \ ۰ \leqslant R \leqslant ۱۰۰ \qquad (۴\text{ـ}۱۲)$$

گراف G یک گراف دو بخشی حاصل از گره‌های U و V می‌باشد که توسط مجموعه یال‌های E به یکدیگر متصل شده‌اند. برای نگاشت گره‌های مجموعه‌ی V به مجموعه‌ی U از حل مسئله‌ی تخصیص

[۳۰] Matched Classes

الگوریتم ۳ نگاشت کلاس‌های کتابخانه و برنامک

ورودی: لیست کلاس‌های کتابخانه‌ای برنامک شامل Cls_{lib} ، لیست کلاس‌های برنامک شامل Cls_{apps}

خروجی: لیستی از دوتایی‌های $(LibraryClass, AppClass)$

۱: قرار بده $f = \emptyset, i = \emptyset$

۲: تا وقتی $Cls_{lib} \neq \emptyset$:

۳: قرار بده $S_{lib,i} = FilterClassByLevel(S_{lib}, i)$

۴: قرار بده $S_{lib} = S_{lib} - S_{lib,i}$

۵: قرار بده $U = \emptyset, V = \emptyset, E = \emptyset$

۶: به ازای $cls \in S_{lib,i}$:

۷: قرار بده $Candidate_{cls} = FindCandidates(cls, Cls_{app})$

۸: قرار بده $M_{cls} = \emptyset$

۹: به ازای $c \in Candidate_{cls}$:

۱۰: اگر $FhashCompare(Sig_c, Sig_{cls}) > Thr_s$:

۱۱: قرار بده $Matched_{cls} = Matched_{cls} \cup c$

۱۲: قرار بده $E = E \cup (c, cls)$

۱۳: اگر $M_{cls} \neq \emptyset$:

۱۴: قرار بده $U = U \cup \{cls\}$

۱۵: قرار بده $V = V \cup M_{cls}$

۱۶: قرار بده $f_i = AssignmentSolver(U, V, E)$

۱۷: قرار بده $f = f \cup f_i$

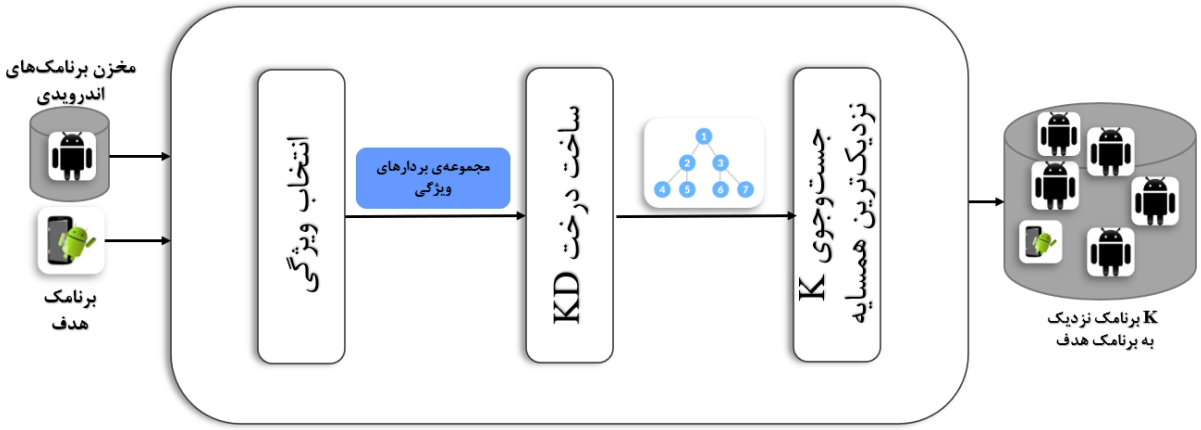
۱۸: قرار بده $i = i + ۱$

۱۹: برگردان f

برای سطح i، در گراف G استفاده شده‌است. خروجی ماژول نگاشت کتابخانه‌ها، تابع یک‌به‌یک و پوشا خواهد بود. در نهایت اجتماع توابع یک‌به‌یک و پوشای به‌دست آمده به ازای هر یک از کلاس‌های کتابخانه‌ای، خروجی ماژول نگاشت می‌باشد که همان نگاشت میان کلاس‌های کتابخانه‌ای و کلاس‌های برنامک می‌باشد. شبه‌کد رویه‌ی نگاشت میان کلاس‌های کتابخانه‌ای و کلاس‌های برنامک را می‌توان در الگوریتم ۳ مشاهده نمود.

۴ـ۳ـ۲ مؤلفه‌ی یافتن نزدیک‌ترین همسایه

جهت افزایش سرعت تشخیص برنامک‌های بازبسته‌بندی شده، از یک مرحله‌ی پیش‌پردازش شامل الگوریتم k نزدیک‌ترین همسایه[۳۱] استفاده شده‌است. همانطور که در شکل ۴ـ۳ مشاهده می‌شود به جهت اجرای الگوریتم نزدیک‌ترین همسایه، ابتدا ۵ ویژگی، مبتنی بر منابع برنامک با استفاده از ماژول استخراج ویژگی، با استفاده از الگوریتم ۴ استخراج می‌شود و درخت ۵ بعدی حاصل از اجرای الگوریتم ۵ ساخته می‌شود. در نهایت با استفاده از اجرای رویه‌ی یافتن نزدیک‌ترین همسایه‌ی در الگوریتم ۶، برنامک‌های کاندید جهت مقایسه‌ی دودویی را انتخاب می‌کنیم.



شکل ۴ـ۳: نمای کلی مؤلفه‌ی یافتن نزدیک‌ترین همسایه

انتخاب ویژگی

به جهت اجرای فرایند یافتن نزدیک‌ترین همسایه‌های هر برنامک، نیازمند مجموعه‌ای از ویژگی‌های هر برنامک هستیم که مقاومت بالایی مقابل راهکارهای مبهم‌نگاری داشته‌باشد. از آنجا که در قسمت مقایسه‌ی دودویی و تشکیل امضای برنامک، بیشتر از ویژگی‌های مبتنی بر کد برنامک‌های اندرویدی استفاده شده‌است،

[۳۱] K-Nearest Neighbors

در این قسمت برای تمایز بهتر برنامک‌ها از ویژگی‌های مبتنی بر منابع برنامک که واسط کاربری آن‌را تشکیل می‌دهد استفاده کرده‌ایم. در ادامه به بررسی و توضیح دلیل انتخاب هر کدام از ویژگی‌های استخراج‌شده در این قسمت می‌پردازیم.

- **تعداد فعالیت‌ها**: هر فعالیت[۳۲] یک صفحه‌ی جداگانه را در برنامک‌های اندرویدی به منظور پیاده‌سازی واسط کاربری در اختیار توسعه‌دهندگان قرار می‌دهد. از آن‌جایی که پیاده‌سازی فعالیت‌های جدید، نیازمند افزودن آن‌ها در حلقه‌ی فراخوانی فعالیت‌هاست، بنابراین، برنامک‌های بازبسته‌بندی شده عموماً دارای تعداد فعالیت مشخص و مشابه هستند چرا که بخشی از اهداف آن‌ها در صورتی محقق می‌شود که کاربران به جهت شباهت‌های واسط کاربری دچار خطا شوند. برای استخراج فعالیت‌های هر برنامک، از فایل $AndroidManifest.xml$ استفاده شده‌است. همانطور که در الگوریتم ۴ مشاهده می‌شود، فعالیت‌های اندرویدی در فایل $AndroidManifest.xml$ با تگ مشخص $Activity$ مشخص شده‌اند. نمونه‌ای از تعریف یک فعالیت در تصویر ۴ـ۴ مشاهده می‌شود.

```
<manifest ... >
  <application ... >
      <activity android:name=".ExampleActivity" />
      ...
  </application ... >
  ...
</manifest >
```

شکل ۴ـ۴: نمونه‌ای از تعریف یک فعالیت

- **تعداد مجوزهای دسترسی برنامک**: ویژگی دیگری که برای طبقه‌بندی از آن استفاده شده‌است، تعداد مجوزهای دسترسی[۳۳] هر برنامک است که در طی اجرای برنامک از کاربر درخواست می‌شود. از آن‌جایی که اجرای بدافزارهای بازبسته‌بندی شده، نیازمند دسترسی‌های گوناگون است بنابراین بدافزارنویسان برای اهداف خود از مجوزهای دسترسی به کرات استفاده می‌کنند. همانند تعداد فعالیت‌های برنامک، برای شمارش تعداد مجوزهای دسترسی، نیازمند شمارش تگ مخصوص $uses-permission$ در فایل فراداده‌ی[۳۴] $AndroidManifest.xml$ هستیم. نمونه‌ای از تعریف حق‌دسترسی جهت استفاده از دوربین کاربر را در تصویر ۴ـ۵ مشاهده می‌کنید.

---

[۳۲] Activity
[۳۳] permissions
[۳۴] Meta-Data

```
<manifest ...>
    <uses-permission android:name="android.permission.CAMERA"/>
    <application ...>
        ...
    </application>
</manifest>
```

شکل ۴ـ۵: نمونه‌ای از تعریف یک حق‌دسترسی

- **تعداد فیلترهای تصمیم**: جابه‌جایی میان صفحات برنامک و یا جابه‌جایی میان صفحات برنامک‌های اندرویدی، با استفاده از فیلترهای تصمیم[۳۵] صورت می‌گیرد. با استفاده از فیلترهای تصمیم برنامک مولفه‌ی[۳۶] درخواستی خود را به سیستم‌عامل اعلام می‌کند. بنابراین رفتار برنامک‌های اندرویدی، از جهتی مبتنی بر فیلترهای تصمیم هستند چرا که عمده‌ی فعالیت‌های یک برنامک مبتنی بر جابه‌جایی میان صفحات مختلف (فعالیت‌ها) با استفاده از فیلترهای تصمیم می‌باشد. در این قسمت تمامی فیلترهای تصمیم ضمنی و یا صریح، با استفاده از فایل $AndroidManifest.xml$ و تگ مخصوص $intent-filter$ استخراج می‌شود.

- **میانگین تعداد فایل‌های png**: برای محاسبه‌ی میانگین تعداد فایل‌های تصویری که پسوند png دارند، از شمارش تعداد پوشه‌های $drawable$ که حاوی تصاویر برنامک هستند استفاده شده‌است. منظور از میانگین در این ویژگی، محاسبه‌ی تقسیم تعداد تمامی فایل‌های png بر تعداد پوشه‌های $drawable$ می‌باشد.

- **میانگین تعداد فایل‌های xml**: همانطور که در الگوریتم ۴ مشاهده می‌شود، برای محاسبه‌ی میانگین تعداد فایل‌های با پسوند xml از محاسبه‌ی تعداد پوشه‌های موجود در پوشه‌ی منابع برنامک یعنی $/res$ استفاده شده‌است. از آن‌جایی که عناصر موجود در واسط‌کاربری برنامک توسط فایل‌های xml پیاده‌سازی می‌شود، بنابراین حذف و یا اضافه‌کردن آن‌ها نیازمند صرف هزینه می‌باشد. دلیل اسفاده از میانگین در دو ویژگی اخیر، ناشی از عدم تغییر این عدد به ازای اضافه کردن پوشه‌های جدید است. برای مثال در صورتی که یک برنامک با استفاده از یک زبان دوم بازبسته‌بندی شود، یک پوشه شامل منابع آن ایجاد می‌شود. حال در این صورت تعداد کل منابع xml اضافه شده‌است اما میانگین آن‌ها تغییر چندانی نمی‌کند.

پس از جمع‌آوری ویژگی‌های ذکرشده، با فرض آن‌که $\vartheta = \{\alpha^i | i \in [۱..N]\}$ مجموعه‌ی برنامک‌های

[۳۵] Intent Filters
[۳۶] Component

موجود در مخزن باشد، بردار ویژگی هر برنامک را با $Features_i = (f_1^i, ..., f_5^i)$ که به ترتیب شامل تعداد فعالیت‌ها، تعداد دسترسی‌های درخواستی، تعداد فیلترهای تصمیم، میانگین تعداد فایل‌های xml و میانگین تعداد فایل‌های png نشان می‌دهیم. پس از ایجاد بردار ویژگی $Features_i$، به منظور هم‌سانی داده‌های آزمون، نرمال‌سازی با استفاده از تابع $Normalized$ روی داده‌ها صورت می‌گیرد. معادله‌ی تابع $Normalized$ به ازای هر ویژگی f_j^i زیر مشخص می‌شود.

$$Normalized(f_j^i) = \sqrt{\frac{f_j^i - min(f_j^{1,..,N})}{max(f_j^{1,..,N}) - min(f_j^{1,..,N})}} \qquad (۱۳-۴)$$

ماژول اسخراج ویژگی‌های مبتنی بر منابع به عنوان ورودی برنامک‌های موجود در مخزن و فایل فرادادهی $AndroidManifest.xml$ و به عنوان خروجی، لیستی از ویژگی‌های ۵ تایی ذکرشده برای هر برنامک بازمی‌گرداند.

ساخت درخت KD

در ادامه برای طبقه‌بندی داده‌های هر برنامک از یک طبقه‌بند مبتنی بر نزدیک‌ترین همسایه استفاده شده‌است. به صورت کلی دو روش پیاده‌سازی برای جست‌وجوی نزدیک‌ترین همسایه وجود دارد. در بدیهی‌ترین حالت ابتدا فاصله‌ی یک نقطه از تمامی نقاط دیگر را محاسبه، در ادامه با استفاده از مرتب‌سازی مبتنی بر فاصله، تعدادی از نزدیک‌ترین همسایه‌ها را استخراج می‌کنیم. همانطور که مشخص‌است روش مورد نظر نیازمند محاسبه‌ی فاصله‌ی نقطه‌ی ورودی با تمامی نقاط دیگر است. در روش دیگری که به KD_KNN شهرت دارد، ابتدا تمامی داده‌های برنامک را در یک داده‌ساختار[۳۷] درختی ذخیره و برای جست‌وجوی K تا از نزدیک‌ترین همسایه‌های هر گره، از آن استفاده می‌کنیم. در این ساختار درختی برای جست‌وجوی گره‌های نزدیک، تمامی داده‌های آزمون به بلوک‌های نزدیک به هم تقسیم و پردازش گره‌های نزدیک تنها در این بلاک صورت می‌گیرد. در ادامه برای درک بهتر الگوریتم جست‌وجوی درختی KNN مثالی از یک مجموعه داده‌ی آزمون دو بعدی را در الگوریتم ۵ بررسی خواهیم کرد.

پس از اجرای الگوریتم ابتدا نقاط ورودی بر اساس عنصر اول دوتایی‌های ورودی جداسازی و مرتب می‌شوند.در ادامه میانه‌ی[۳۸] داده‌های مرتب‌شده‌ی حاصل از مرحله‌ی قبل، محاسبه و عناصر داده‌ای بیش از مقدار میانه در سمت راست درخت و عناصر کم‌تر در سمت چپ درخت تقسیم می‌شوند. این مرحله، در واقع سازوکار ایجاد بلوک‌های داده‌ای انجام می‌شود. از آن‌جایی که فاصله‌ی دو نقطه در فضا از یکدیگر با

[۳۷] Data Structure
[۳۸] Median

استفاده از معادله‌ی ۴ـ۱۴ محاسبه می‌شود، بنابراین جداسازی بلوک‌های داده‌ای در الگوریتم ۵ به صورت یکی در میان، بر اساس عناصر اول و دوم از لیست داده‌ای تقسیم می‌شوند.

$$d(x,y) = \sqrt{(x_۲ - x_۱)^۲ + (y_۲ - y_۱)^۲} \qquad (۱۴-۴)$$

برای ساخت درخت مبتنی بر ویژگی‌های استخراج‌شده از الگوریتم ۴، از بردار ویژگی حاصل استفاده کرده

الگوریتم ۴ استخراج ویژگی‌های مبتنی بر منابع

ورودی: برنامک‌های موجود در مخزن $DsApps$ و فایل فراداده‌ی هرکدام $Manifest_{app}$

خروجی: ویژگی‌های ۵ تایی از هر برنامک شامل مجموعه‌ی $Features$

۱: قرار بده $Features = \emptyset$

۲: به ازای $app \in DsApps$:

۳: به ازای $tag \in Manifest_{app}$:

۴: اگر $tag = Activity$:

۵: قرار بده $Features_{app,AcNumber} = Features_{app,AcNumber} + ۱$

۶: اگر $tag = uses - premession$:

۷: قرار بده $Features_{app,PNumber} = Features_{app,PNumber} + ۱$

۸: اگر $tag = intent - filter$:

۹: قرار بده $Features_{app,IfNumber} = Features_{app,IfNumber} + ۱$

۱۰: قراربده $\#TotalDrawableDir = GetTotalDrNum()$

۱۱: قرار بده $\#TotalPngFiles = GetTotalPngs("/res")$

۱۲: قرار بده $Features_{app,AvgPngPerDir} = \frac{\#TotalPngFiles}{\#GetTotalDrNum}$

۱۳: قراربده $\#TotalResDir = GetTotalDirNum()$

۱۴: قرار بده $\#TotalXmlFiles = GetTotalXmls("/res")$

۱۵: قرار بده $Features_{app,AvgXmlPerDir} = \frac{\#TotalXmlFiles}{\#GetTotalDirNum}$

۱۶: برگردان $Features$

و در نهایت درخت ۵ بعدی حاوی اطلاعات برنامک‌های اندرویدی ساخته می‌شود. هر گره درخت مذکور حاوی یک بردار ویژگی ۵ تایی است و مطابق الگوریتم ۵ تصمیم‌گیری در مورد تقسیم داده‌های موجود در هر سطح با توجه به یکی از عناصر ۵ تایی صورت می‌گیرد. به عنوان مثال، زمانی که در سطح اول و ریشه‌ی درخت هستیم، عناصر اول ۵تایی‌های مجموعه‌داده را جدا و آن‌ها را مرتب می‌کنیم. سپس با

توجه به محاسبه‌ی میانه مقدار $Median$ را برای این گروه داده‌ای محاسبه کرده و ۵ تایی مربوط به عنصر $Median$ در ریشه قرار می‌گیرد. سپس باقی مجموعه‌ی داده‌ای با توجه به بزرگ‌تر بودن و یا کوچک‌تر بودن از عنصر $axis$ که همان شاخص تصمیم است، تقسیم می‌شوند و این رویه به صورت بازگشتی برای تمامی بلوک‌های داده‌ای و گره‌های درخت تکرار می‌شود تا زمانی که درخت ساخته شود.

ماژول ساخت درخت KD در این پژوهش لیستی از عناصر ۵ تایی حاوی ویژگی‌های مبتنی بر منابع و به طول n می‌باشد. خروجی ماژول یک درخت ۵ بعدی حاوی ویژگی‌های مذکور است که می‌توان رویه‌ی جست‌وجوی نزدیک‌ترین همسایه مطابق الگوریتم ۶ را از مرتبه‌ی $log(n)$ اجرا کرد.

الگوریتم ۵ ساخت درخت دوبعدی جست‌وجوی k نزدیک‌ترین همسایه

ورودی: لیستی از داده‌های دوتایی $Points$ به صورت (x, y) و به طول n

ورودی: عمق سطح $Depth$

خروجی: درخت دوبعدی شامل داده‌های ورودی

۱: قرار بده $Dimension = Depth \mod ۲$

۲: اگر $Dimension = ۱$:

۳: قرار بده $Median = GetMedian(Sort(x_۱, ..., x_n))$

۴: در غیر این صورت:

۵: قرار بده $Median = GetMedian(Sort(y_۱, ..., y_n))$

۶: قرار بده $Node.data = Median$

۷: قرار بده $Node.axis = Dimension$

۸: قرار بده $Node.leftChild = kdtree(points\ \ in\ \ points\ \ before\ \ median, depth + ۱)$

۹: قرار بده $Node.rightChild = kdtree(points\ \ in\ \ points\ \ after\ \ median, depth + ۱)$

جست‌وجوی k همسایه‌ی نزدیک

پس از ساخت داده‌ساختار درختی پنج‌بعدی در مرحله‌ی قبل، حال هدف یافتن نزدیک‌ترین همسایه‌های هر برنامک به جهت بررسی دوتایی آن‌ها می‌باشد. به همین جهت در این قسمت از الگوریتم جست‌وجوی نزدیک‌ترین همسایه که در پژوهش [۹۶] معرفی شده‌است، استفاده کرده‌ایم. شبه‌کد این رویه را می‌توان در الگوریتم ۶ مشاهده نمود.

الگوریتم ۶ جست‌وجوی نزدیک‌ترین همسایه

ورودی: ریشه‌ی درخت $Root$

ورودی: نقطه‌ی آزمون $TPoint$

خروجی: نزدیک‌ترین نقطه به ورودی $NearestPoint$

۱: قرار بده $Path = \emptyset$

۲: تا وقتی $Root! = leaf$:

۳: $Path.add(Root)$

۴: اگر $Root[data][Root.axis] > TPoint[axis]$:

۵: $Searchtree(rightChild)$

۶: در غیر این صورت:

۷: $Searchtree(leftChild)$

۸: به ازای $Point \in Path$:

۹: قراربده $NearestDist = Distance(Point, TPoint)$

۱۰: اگر $|Point[Data][axis] - TPoint[axis]| > |TPoint[axis] - Root[Data][axis]|$:

۱۱: $Travel(Root.nextchild)$

۱۲: قرار بده $Dist = Distance(ChildPoint, TPoint)$

۱۳: اگر $Dist > NearestDist$:

۱۴: قرار بده $NearestDist = Dist$

۱۵: قرار بده $NPoint = Point$

۱۶: قرار بده $NearestPoint = NPoint$

۱۷: برگردان $NearestPoint$

ورودی الگوریتم شامل ریشه‌ی درخت KD و هدف آن یافتن نزدیک‌ترین نقطه از میان نقاط درون درخت به نقطه‌ی ورودی $Tpoint$ می‌باشد. به طور کلی الگوریتم مذکور را می‌توان به دو قسمت تقسیم کرد، در قسمت اول در طی یک روند رو به جلو سعی می‌کنیم با توجه به نقطه‌ی ورودی، مکان درست آن را در برگ درخت بیابیم. به همین منظور از یک ساختار مبتنی بر درخت تصمیم[۳۹] استفاده می‌کنیم. در این ساختار، در صورتی که مقدار شاخص تصمیم[۴۰] $axis$ بردار ورودی، از همین مقدار در بردار گره بیشتر باشد، آن‌گاه جست‌وجو را در زیردرخت سمت راست ریشه انجام می‌دهیم و در صورتی که مقدار آن کوچکتر باشد آن‌گاه زیر درخت سمت چپ را بررسی می‌کنیم. این روند بازگشتی را تا زمانی که به گره برگ برسیم ادامه می‌دهیم و مسیر طی‌شده را ذخیره می‌کنیم. در مرحله‌ی بعدی با توجه به نقاط درون مسیر $Path$ تمامی گره‌های مسیر را به صورت عقب‌گرد[۴۱] بررسی و نزدیک‌ترین بردار ویژگی[۴۲] به گره ورودی را پیدا کرده و آن را بازمی‌گردانیم. متد $Distance$ در الگوریتم ۶ فاصله‌ی میان دو بردار ۵ تایی را با استفاده از فاصله‌ی اقلیدوسی به صورت زیر محاسبه می‌کند.

$$d(f^1, f^2) = \sqrt{(f^1_1 - f^2_1)^2 + (f^1_2 - f^2_2)^2 + ... + (f^1_5 - f^2_5)^2} \qquad (۱۵-۴)$$

پس از ایجاد درخت و پیاده‌سازی الگوریتم نزدیک‌ترین همسایه، تعداد Kفراخوانی از الگوریتم ۶ منجر به دریافت لیستی از K نزدیک‌ترین همسایه‌های بردار ورودی می‌شود

۴-۳-۳ مقایسه‌ی دودویی و تشخیص برنامک‌های بازبسته‌بندی شده

هدف از پیاده‌سازی مرحله‌ی طبقه‌بندی، کاهش فضای مقایسه‌ی دودویی در این مرحله بوده‌است که منجر به افزایش سرعت مقایسه می‌شود. پس از اجرای الگوریتم یافتن K تا از نزدیک‌ترین همسایه‌های برنامک ورودی با استفاده از ویژگی‌های مبتنی بر منبع، در این قسمت مقایسه‌ی دودویی میان K برنامک و برنامک ورودی صورت می‌گیرد.

پس از شناسایی کلاس‌های کتابخانه‌ای در الگوریتم ۳، نگاشتی از کلاس‌های هر برنامک داخل مخزن و برنامک هدف با کلاس‌های کتابخانه‌ای به صورت $F : LibraryClasses \rightarrow AppClasses$ به دست می‌آید. در ادامه امضای هر کدام از کلاس‌های برنامک را تشکیل می‌دهیم. امضای هر برنامک از الحاق تمامی امضای کلاس‌های آن ایجاد می‌شود. سپس شروع به حذف کلاس‌های کتابخانه‌ای با توجه به نگاشت F مطابق رویه‌ی الگوریتم ۷ از هر برنامک می‌کنیم و مجموعه‌ی کلاس‌های برنامک با حذف کتابخانه‌های اندرویدی تشکیل مجموعه‌ی S'_{app} را می‌دهند که به صورت زیر محاسبه شده‌است.

[۳۹] Decision Tree
[۴۰] Decision Index
[۴۱] Back Track
[۴۲] Feature Vector

$AppClasses$ مجموعه‌ی تمامی کلاس‌های کتابخانه‌ای است که در نگاشت F به دست آمده‌است.

$$Cls'_{app} = Cls_{app} - AppClasses \qquad (۱۶-۴)$$

الگوریتم ۷ تشخیص جفت بازبسته‌بندی شده

ورودی: مجموعه امضای کلاس‌های دو برنامک مورد مقایسه شامل $Cls_{app۱}, Cls_{app۲}$

ورودی: مجموعه امضای کلاس‌های کتابخانه‌ای حاصل از الگوریتم ۳ برای دو برنامک ورودی شامل $Appclasses_۱$ و $AppClasses_۲$

خروجی: متغیر $IsRepackagedPair$ ◁ *در صورتی که متغیر* $True$ *باشد دو برنامک بازبسته‌بندی شده و در غیر این صورت جفت ورودی بازبسته‌بندی یکدیگر نیستند*

۱: قرار بده $Cls'_{app۱} = Cls_{app۱} - AppClasses_۱$

۲: قرار بده $Cls'_{app۲} = Cls_{app۲} - AppClasses_۲$

۳: قرار بده $Score = FhashCompare(Cls'_{app۱}, Cls'_{app۲})$

۴: اگر $Score \geqslant Thr_c$:

۵: قرار بده $IsRepackagedPair = True$

۶: در غیر این صورت:

۷: قرار بده $IsRepackagedPair = False$

۸: برگردان $IsRepackagedPair$

پس از حذف کلاس‌های کتابخانه‌ای از هر امضا، در این مرحله امضای اصلی برنامک حاوی کلاس‌هایی که توسط توسعه‌دهنده پیاده‌سازی شده‌اند را با یکدیگر مقایسه می‌کنیم. مقایسه‌ی امضای برنامک‌ها با استفاده از روش‌های چکیده‌سازی محلی صورت می‌گیرد. استفاده از روش‌های چکیده‌سازی معمولی، نظیر $MD۵$ برای تشابه‌سنجی، منجر به افزایش خطا در صورت مبهم‌نگاری برنامک‌ها خواهد شد چرا که روش‌های معمول عموماً برای تولید شناسه‌ی یکتا[۴۳] کاربرد دارند و در صورتی که قسمت کوچکی از ورودی آن‌ها تغییر کند، آنگاه چکیده‌ی جدید حاصل از این توابع، به صورت کامل متفاوت خواهد بود. بنابراین استفاده از این روش‌ها برای شباهت‌سنجی میان امضای برنامک‌های اندرویدی پیشنهاد نمی‌شود. به همین منظور، در شباهت‌سنجی میان فایل‌های متنی و یا به جهت تشخیص تکرار ساختارهای واحد در فایل‌های مشابه، از روش‌های مبتنی بر چکیده‌سازی محلی استفاده می‌شود. ساختار کلی این روش‌ها، استفاده از تشابه میان بلوک‌های تکراری در متون می‌باشد و به همین دلیل، اگر دو فایل مشابه به عنوان ورودی به آن‌ها

[۴۳] Unique

داده‌شود، آنگاه چکیده‌ی نهایی نیز به همان میزان مشابه خواهد بود. در صورتی که میزان تشابه میان امضای نهایی برنامک‌های موجود در مخزن از یک حد آستانه مانند Thr_c کم‌تر باشد، آنگاه دو برنامک مورد نظر بازبسته‌بندی شده تشخیص داده می‌شوند.

فصل ۵

ارزیابی

روش پیاده‌سازی شده در این پژوهش را می‌توان از نظر ساختار آن، در دسته‌ی پژوهش‌های مبتنی بر تحلیل ایستا قرار داد. یکی از آخرین پژوهش‌های موجود در این دسته، روشی است که توسط آقای ترکی[۵] مبتنی بر تحلیل گراف فراخوانی کلاسی[۱] و استخراج امضا از کلاس‌های برنامک ارائه شده‌است. پژوهش آقای ترکی[۵] در ادامه‌ی پژوهش آقای وانگ[۵۳] و به جهت بهبود دقت آن انجام شده‌است. از آن‌جایی که هدف این پژوهش افزایش کارایی پژوهش‌های مرتبط با تشخیص برنامک‌های بازبسته‌بندی شده می‌باشد، بنابراین در این فصل روش پیشنهادی را از دو منظر سرعت تشخیص و معیارهای دقت[۲] و فراخوان[۳] بررسی خواهیم کرد و قسمتی از پژوهش را به مقایسه‌ی راهکار پیشنهادی این پژوهش با آخرین روش مشابه اختصاص خواهیم‌داد. در ادامه‌ی ارزیابی، معیارهای دقت و فراخوان را به صورت زیر تعریف می‌کنیم. منظور از TP, FP, FN به ترتیب تعداد منفی غلط[۴]، مثبت غلط [۵]و مثبت‌صحیح[۶]، می‌باشد. منظور از منفی‌غلط، حالتی است که در آن جفت‌بازبسته‌بندی شده به اشتباه تشخیص داده نمی‌شود. مثبت‌غلط، حاصل از تشخیص بازبسته‌بندی برای یک جفت بازبسته‌بندی نشده و مثبت‌صحیح، زمانی است که ما جفت بازبسته‌بندی شده را به درستی تشخیص می‌دهیم.

$$Precision = \frac{TP}{TP + FP} \quad (۱-۵)$$

$$Recall = \frac{TP}{TP + FN} \quad (۲-۵)$$

[۱] Class Call Graph
[۲] Precision
[۳] Recall
[۴] False Negative
[۵] False Positive
[۶] True Positive

در ادامه‌ی این فصل ابتدا پارامترهای ثابت موجود در الگوریتم‌های ارائه‌شده در این پژوهش را عنوان و مقدار آن‌ها را به ازای ارزیابی‌های انجام‌شده، اعلام می‌کنیم. سپس مجموعه‌ی داده‌ای مورد ارزیابی در این پژوهش و ویژگی‌های آن را بررسی خواهیم‌کرد. سپس ارزیابی راهکار پیشنهادی پژوهش را از دو دیدگاه انجام می‌دهیم. در قسمت اول، ابتدا معیارهای ارزیابی را در مقایسه‌ی دودویی برنامک‌های اندرویدی، بر روی مجموعه‌ی داده[۷] بررسی می‌کنیم. اجرای ارزیابی با توجه به ایده‌ی مطروحه در مرحله‌ی تشکیل امضا و مقایسه‌ی آن با پژوهش پیشین انجام می‌گیرد. در قسمت دوم، ارزیابی پژوهش با استفاده از افزودن الگوریتم طبقه‌بندی نزدیک‌ترین همسایه انجام می‌گیرد و مقایسه‌ی دودویی با استفاده از کاهش فضای مقایسه، به عنوان ایده‌ی اصلی این پژوهش مطرح می‌گردد.

۵ـ۱ پارامترهای پژوهش

در این قسمت پارامترهای ثابت استفاده شده در رویه‌های مشروح در فصل ۴ را بررسی و مقدار هر کدام را برای اجرای ارزیابی پژوهش را عنوان می‌کنیم.

۵ـ۱ـ۱ مولفه‌ی تشخیص کتابخانه‌های اندرویدی

برای تحلیل ایستای برنامک‌های اندرویدی و تشکیل امضای کلاسی، از چارچوب سوت استفاده شده‌است. چارچوب سوت ابزاری مبتنی بر زبان جاوا است که اجازه‌ی تحلیل برنامک‌های اندرویدی و استخراج ویژگی‌های گوناگون کدپایه[۸] از آن را به کاربران می‌دهد. در قسمت تشکیل امضای کلاس‌های کتابخانه‌ای و استخراج ویژگی‌های مذکور در فصل ۴ از این ابزار استفاده شده‌است.
در قسمت مقایسه‌ی امضای کلاسی به منظور یافتن کلاس‌های کاندید و تشکیل نگاشت میان کلاس‌های کتابخانه‌ای و کلاس‌های برنامک و همچنین مقایسه‌ی امضای نهایی برنامک‌های اندرویدی، از روش‌های چکیده‌سازی محلی استفاده شده‌است. به جهت ارزیابی روش پژوهش، از سه شمای[۹] چکیده‌سازی محلی، شامل روش‌های $TLSH, SSdeep$ و $Sdhash$ استفاده شده‌است. در قسمت ارزیابی مقایسه‌ی دودویی، به تفضیل به مقایسه‌ی دقت و زمان محاسبه‌ی چکیده در این سه روش خواهیم پرداخت.
در ادامه، پارامترهای ثابت استفاده‌شده در مؤلفه‌ی تشخیص کدهای کتابخانه‌ای را بررسی و مقدار ثابت آن‌ها را توضیح می‌دهیم.

---
[۷] Data Set
[۸] Code Base
[۹] schema

- **پارامتر** Thr_L: پارامتر مذکور به عنوان پارامتر ثابت در ماژول فیلتر طول امضا، به‌جهت فیلتر کلاس‌های کاندید مبتنی بر طول کلاس هدف استفاده شده‌است. مقدار این پارامتر بر اساس طول کلاس هدف و بر اساس ضریبی از آن محاسبه می‌شود که روال محاسبه‌ی آن در الگوریتم ۸ مشاهده می‌شود.

الگوریتم ۸ محاسبه‌ی T_L

ورودی: کلاس هدف در فیلتر طولی T_C

خروجی: مقدار T_L مبتنی بر طول امضای کلاس هدف

۱: قرار بده $L = length(Sig_{Tc})$

۲: اگر $L \leqslant ۱۰۰۰$:

۳: $T_L = L \times ۰/۶$

۴: در غیر این صورت: اگر $L \leqslant ۵۰۰۰$:

۵: $T_L = L \times ۰/۵$

۶: در غیر این صورت: اگر $L \leqslant ۱۰۰۰۰$:

۷: $T_L = L \times ۰/۴$

۸: در غیر این صورت: اگر $L \leqslant ۵۰۰۰۰$:

۹: $T_L = L \times ۰/۳$

۱۰: در غیر این صورت: اگر $L \leqslant ۱۰۰۰۰۰$:

۱۱: $T_L = L \times ۰/۲$

۱۲: در غیر این صورت:

۱۳: $T_L = L \times ۰/۲$

۱۴: برگردان T_L

- **پارامتر** Thr_F: مقدار این پارامتر در ماژول فیلتر کتابخانه‌ها به عنوان حد کمینه برای تشابه میان کلاس‌های کتابخانه‌ای و برنامک استفاده می‌شود که به جهت ارزیابی در این پژوهش مقدار ۷۰ گرفته‌است.

- **پارامتر** Thr_S: این پارامتر به جهت حد کمینه‌ی تشابه میان کلاس‌های کتابخانه‌ای و برنامک به منظور ایجاد نگاشت استفاده می‌شود که مقدار آن برابر ۵۰ قرار داده‌شده است.

- **پارامتر** N: مقدار این پارامتر در ماژول فیلتر کتابخانه‌ها، برای تعیین تعداد کلاس‌های کتابخانه‌ای استفاده می‌شود که برابر ۱ قرار گرفته‌است.

۵ـ۱ـ۲ مؤلفه‌ی یافتن نزدیک‌ترین همسایه

در ماژول استخراج ویژگی به جهت ایجاد بردارهای ویژگی برای هر برنامک، از نرم‌افزار $apktool$ به جهت انجام دیس‌اسمبل فایل‌های apk استفاده شده‌است. $apktool$ ابزار قدرتمندی است که اجازه‌ی انجام عملیات مهندسی معکوس[۱۰] و دستیابی به کداصلی برنامک‌های اندرویدی را ممکن می‌سازد.

- **پارامتر** K: مقدار این پارامتر مشخص‌کننده‌ی تعداد همسایه‌های نزدیک به برنامک هدف می‌باشد که به جهت ارزیابی با توجه به مجموعه‌ی داده‌ی آزمون، مقدار برای ۲۵۰ برای آن در نظر گرفته شده‌است.

۵ـ۱ـ۳ مؤلفه‌ی مقایسه‌ی دودویی و تشخیص برنامک‌های بازبسته‌بندی شده

در این مولفه برای مقایسه‌ی امضای نهایی برنامک‌های اندرویدی با استفاده از چکیده‌سازی محلی، از سه کتابخانه‌ی پایتونی $Tlsh$، $Fuzzyhashlib$، $ssdeep$ استفاده‌شده‌است.

- **پارامتر** Thr_c: این پارامتر حد بیشینه و یا کمینه را با توجه به روش مورد استفاده در قسمت چکیده‌سازی مشخص می‌کند. مقدار حد کمینه برای روش‌های چکیده‌سازی محلی $sdhash$ ، $ssdeep$، عدد ۷۰ و برای روش $tlsh$ مقدار بیشینه‌ی ۶۰ در نظر گرفته‌شده است.

۵ـ۲ مجموعه داده‌ی آزمون

در قسمت تشخیص کدهای کتابخانه‌ای، با استفاده از جمع‌آوری کتابخانه‌های مشهور و استفاده از مخزن کتابخانه‌های پژوهش آقای وانگ[۵۳] (شامل ۴۵۱ کتابخانه) در نهایت مخزنی از ۸۷۷ کتابخانه‌ی اندرویدی ساخته‌شد. مخزن کتابخانه‌های مورد استفاده در پژوهش آقای ترکی [۵]، همان مخزن کتابخانه‌ای در پژوهش آقای وانگ می‌باشد. پس از بررسی و ارزیابی‌های اولیه‌ی استخراج کدهای کتابخانه‌ای در این پژوهش متوجه شدیم که بخشی از کتابخانه‌های اندرویدی شناسایی نمی‌شود. بنابراین با افزایش تعداد کتابخانه‌های مخزن، توانستیم مجموعه داده‌ی آزمون استفاده شده در این پژوهش را بهبود ببخشیم.

[۱۰] Reverse Engineering

به جهت ارزیابی تشخیص برنامک‌های بازبسته‌بندی شده، از داده‌ی آزمون جمع‌آوری شده در پژوهش اندروزو[۱۱] [۹۷] استفاده شده‌است. این مجموعه داده‌ی آزمون شامل ۱۵۰۰۰ جفت بازبسته‌بندی شده است که از مجموعه‌ی ۲۲ میلیون برنامک اندرویدی استخراج شده‌است. برای ساخت داده‌ی آزمون پژوهش، ۷۹۶ جفت بازبسته‌بنده شده و ۴۰۰ جفت برنامک‌های غیر تقلبی از میان این مجموعه استفاده شده‌است. همچنین مجموعه‌ی داده‌ی آزمون پژوهش شامل ۱۱۸۱ برنامک اندرویدی است که جفت‌های تقلبی و غیر تقلبی را تشکیل می‌دهند.

۵ـ۳ ارزیابی و مقایسه

از آن‌جایی که هدف این پژوهش افزایش کارایی تشخیص برنامک‌های اندرویدی بازبسته‌بندی شده‌است. بنابراین در این قسمت، ابتدا مقایسه‌ای از روش‌های چکیده‌سازی محلی از نظر زمان اجرا و دقت تشخیص، به صورت دودویی خواهیم داشت و دلایل بهبود این روش را نسبت به روش پیشین بررسی خواهیم کرد. در قسمت بعدی ایده‌ی اصلی پژوهش یعنی استفاده از طبقه‌بندی نزدیک‌ترین همسایه برای کاهش فضای مقایسه را مورد ارزیابی قرار می‌دهیم.

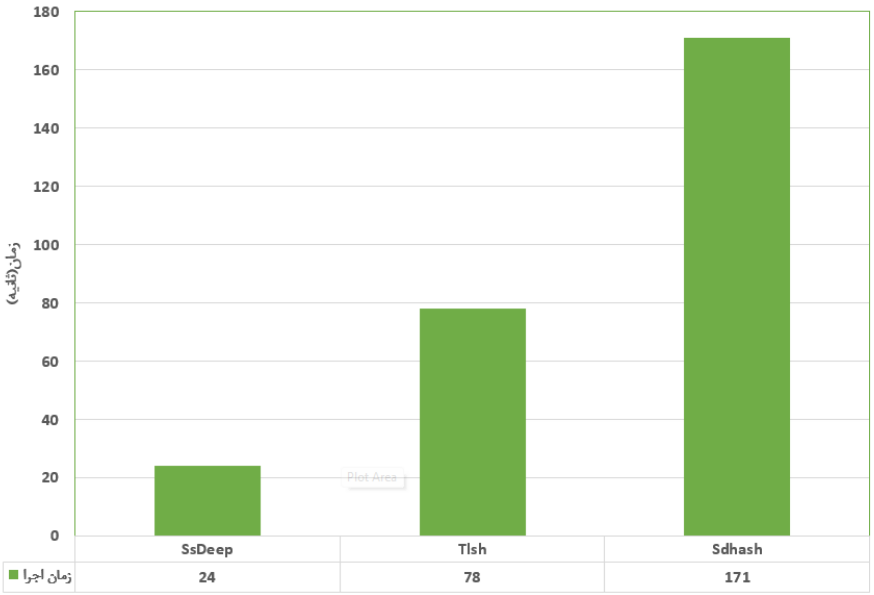
به جهت پیاده‌سازی الگوریتم‌های سه مولفه‌ی تشخیص کدهای کتابخانه‌ای، طبقه‌بندی نزدیک‌ترین همسایه و تشخیص برنامک‌های بازبسته‌بندی از زبان پایتون استفاده‌شده است. همچنین به جهت مقایسه‌ی امضای کلاس‌ها و امضای برنامک‌های اندرویدی از سه کتابخانه‌ی $Tlsh$، $Fuzzyhashlib$، $ssdeep$ در زبان پایتون استفاده شده‌است. ارزیابی‌های پژوهش در ماشینی با پردازنده‌ی $Corei۵ - ۱۰۴۰۰$ و $۳۲G$ حافظه‌ی Ram اجرا شده‌است.

۵ـ۳ـ۱ مقایسه‌ی دودویی بدون طبقه‌بندی

با بررسی روش پیشین پیاده‌سازی شده در پژوهش آقای ترکی، دریافتیم که به صورت کلی این پژوهش را از دو جنبه‌ی متفاوت می‌توان بهبود بخشید. امضای متد در پژوهش آقای ترکی، حاوی مقادیر زیادی افزونگی ناشی از اضافه‌نمودن امضای کلاس در خروجی و ورودی متدها می‌باشد به همین علت طول امضا در برخی از برنامک‌های اندرویدی مخزن از نیم میلیون کاراکتر نیز فراتر رفته‌است. ایجاد افزونگی در امضای متد منجر به افزایش قابل توجه طول امضای کلاسی می‌شود.
به دلیل افزایش افزونگی در امضای کلاسی، تعداد کلاس‌های کاندید ناشی از اعمال الگوریتم‌های ساختاری و طولی افزایش یافته و موجب افزایش گره‌های گراف دوبخشی در حل مسأله‌ی تخصیص در ماژول نگاشت

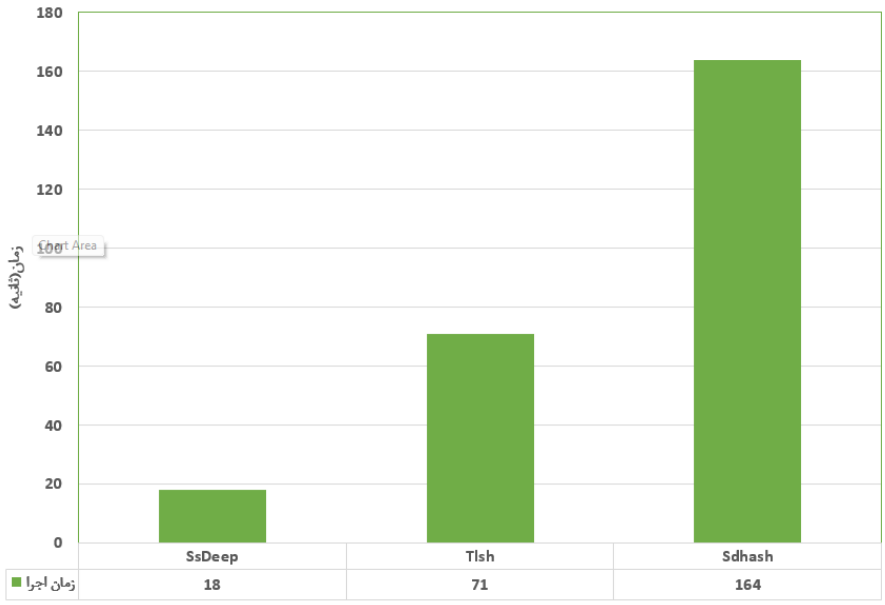
[۱۱] Androzoo

کلاس‌های کتابخانه‌ای و برنامک می‌گردد. از آن‌جایی که حل مسئله‌ی تخصیص در گراف‌های دوبخشی از مرتبه‌ی زمانی n^3 است بنابراین افزایش تعداد گره‌های گراف دوبخشی حاصل منجر به کاهش سرعت قابل توجه در این پژوهش می‌شود. بنابراین در قسمت تشکیل امضا در این پژوهش، توانستیم با اضافه‌نمودن ویژگی‌های ثانویه و مورد نیاز، نظیر توابع فراخوانی‌شده در بدنه‌ی کلاس، تا حدودی افزونگی موجود در این پژوهش را مرتفع سازیم به طوری که ارزیابی حاصل از اعمال امضای جدید، دقت خوبی داشته‌باشد.



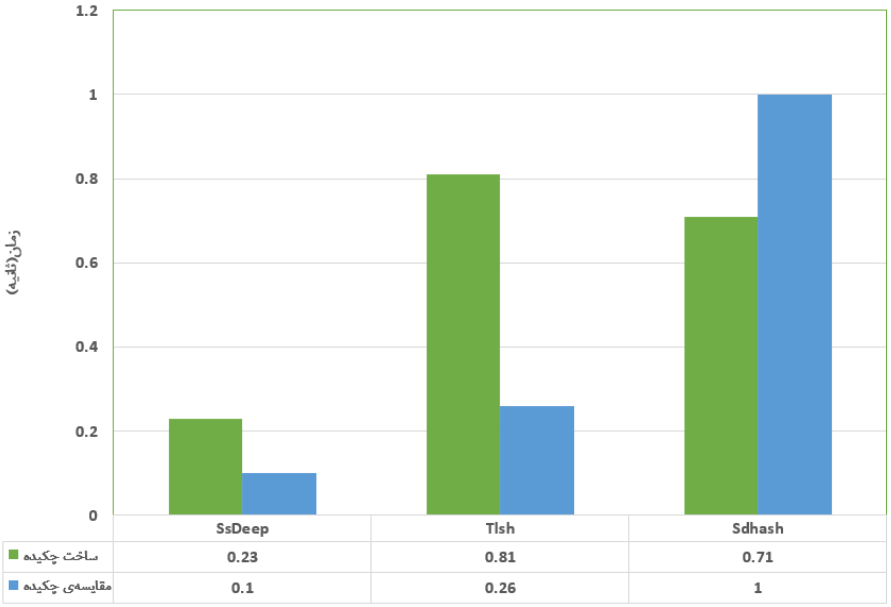
شکل ۵ـ۱: مقایسه‌ی میانگین زمان اجرای روال تشخیص برنامک‌های بازبسته‌بندی شده

همانطور که در شکل ۵ـ۱ مشاهده می‌شود، با مقایسه‌ی میانگین زمان لازم برای اجرای تشخیص بازبسته‌بندی از مرحله‌ی ابتدایی تشخیص کدهای کتابخانه‌ای تا انتها بر روی ۷۹۶ جفت بازبسته‌بندی شده و ۴۰۰ جفت غیر تقلبی، به ازای ۳ روش‌های چکیده‌سازی محلی، می‌توان متوجه شد که روش $ssdeep$ دقت بالاتری نسبت به باقی روش‌ها دارد چرا که میانگین زمان اجرای روال تشخیص بازبسته‌بندی به ازای برنامک‌های داخل مخزن، از تمامی روش‌های دیگر بهتر است. از طرفی با مقایسه‌ی نمودارهای ۵ـ۲ و ۵ـ۴ مشخص می‌شود که عمده‌ی زمان اجرا مختص تشخیص کتابخانه‌های اندرویدی است چرا که در این مرحله از تشخیص، یک مسأله‌ی گرافی با مرتبه‌ی بالایی حل می‌شود. بنابراین می‌توان نتیجه‌گرفت که با بهبود امضای برنامک‌های اندرویدی و در نتیجه حذف افزونگی موجود در پژوهش ترکی، می‌توان سرعت تشخیص را به صورت قابل توجهی افزایش داد چرا که با جذف این افزونگی‌ها، گره‌های حاصل از گراف دوبخشی کاهش یافته و مسأله‌ی گرافی با سرعت بالاتری حل می‌شود.



شکل ۵ـ۲: مقایسه‌ی میانگین زمان اجرای مرحله‌ی تشخیص کدهای کتابخانه‌ای

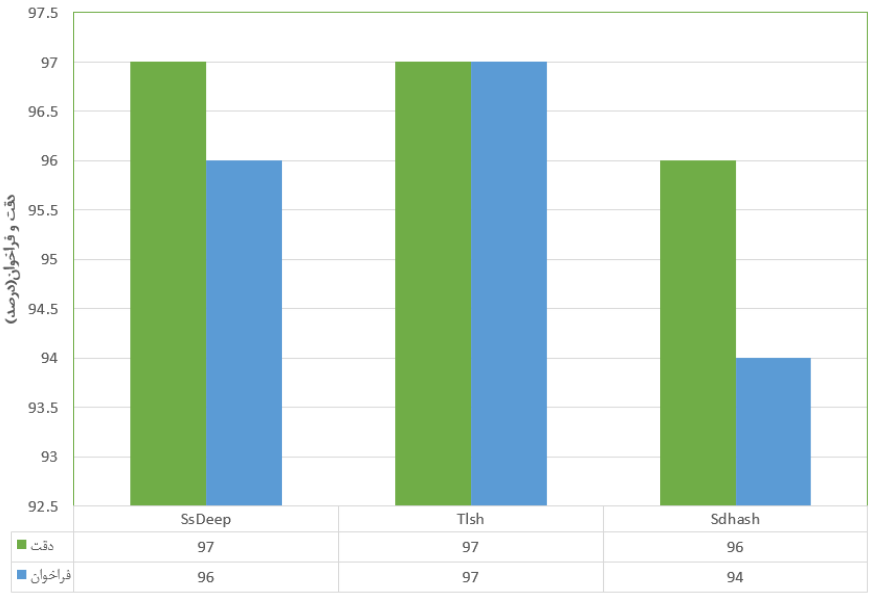
در نهایت با حذف افزونگی‌های امضا در پژوهش ترکی، توانستیم روش موجود را به ازای مقایسه‌ی دودویی برنامک‌های اندرویدی بهبود بخشیم. همانطور که در شکل ۵ـ۵ مشخص‌شده است، استفاده از امضای روش ترکی به دلیل افزونگی زیاد، در مرحله‌ی تشخیص کتابخانه‌های اندرویدی سرعت بسیار پایینی دارد.



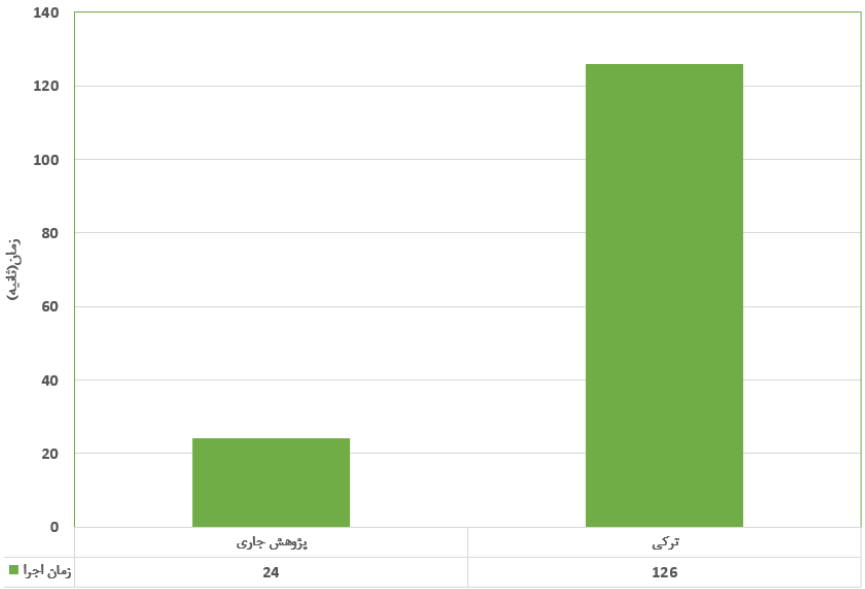
شکل ۵ـ۳: مقایسه‌ی میانگین زمان اجرای مراحل ساخت چکیده و مقایسه‌ی چکیده‌ها

مقایسه‌ی صورت‌گرفته میان روش این پژوهش و پژوهش ترکی با استفاده از تولید تولید امضا و مقایسه‌ی

مبتنی بر چکیده‌سازی $Ssdeep$ انجام شده‌است چرا که این روش، چه از نظر دقت مطابق شکل ر؟جوعشکل: و چه سرعت نتیجه‌ی بهترین نسبت به باقی روش‌ها داشت.با مقایسه‌ی روش پیاده‌سازی شده و روش ترکی متوجه‌شدیم که در بخش تشخیص کتابخانه‌های اندرویدی، به طور میانگین ۳۱۵ گره کم‌تر از مرحله‌ی فیلتر کتابخانه‌ها عبور می‌کنند که این موجب می‌شود تعداد گره‌های گراف دوبخشی نهایی بسیار کمتر از روش آقای ترکی باشد.



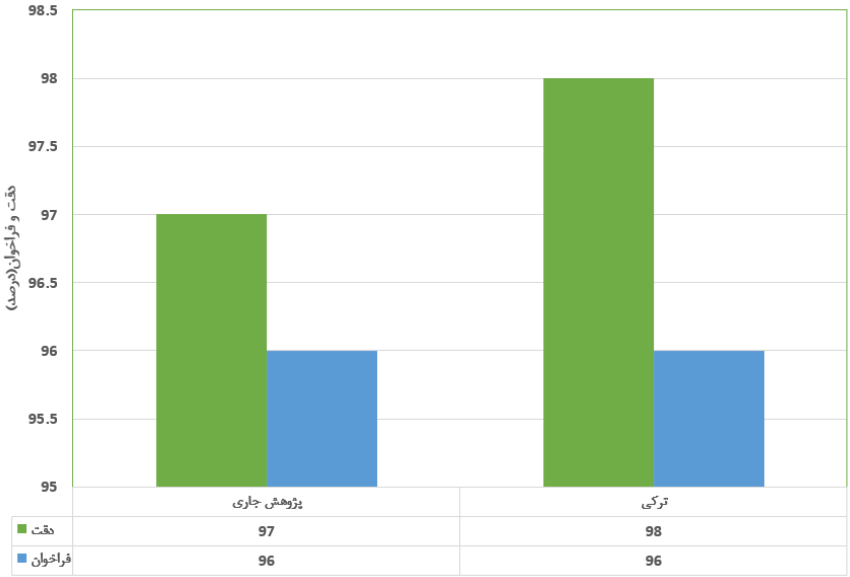
شکل ۵ـ۴: مقایسه‌ی دقت و فراخوان روش‌های چکیده‌سازی محلی در پژوهش جاری



شکل ۵ـ۵: مقایسه‌ی میانگین زمان اجرای مراحل تشخیص پژوهش جاری با پژوهش ترکی[۵]

همچنین در مرحله‌ی ساخت امضای برنامک، به دلیل جست‌وجوی زیاد ناشی از افزونگی آن، تولید امضا در پژوهش جاری به صورت میانگین ۱۰ ثانیه به طور می‌انجامد. این در حالی‌است که با حذف افزونگی‌های امضا توانستیم زمان تشکیل امضای برنامک‌های اندرویدی را به ۴/۷ ثانیه کاهش دهیم.

همانطور که مشاهده‌شد، با حذف افزونگی‌های امضای برنامک در پژوهش آقای ترکی توانستیم سرعت پژوهش اخیر را بهبود ببخشیم. به علاوه در ابتدای این فصل بیان‌کردیم که هدف از این پژوهش افزایش کارایی پژوهش‌های اخیر، با در نظر گرفتن معیارهای سرعت و دقت، بوده‌است. بنابراین قربانی نمودن دقت پژوهش در ازای دریافت سرعت بالا منطقی نمی‌باشد و ایجاد یک توازن میان این دو معیار، حیاتی است. بنابراین با حذف افزونگی‌های پژوهش ترکی، نیازمند اضافه‌نمودن ویژگی‌هایی بودیم که در کنار یکتابودن در کلاس‌های برنامک، بتواند بخشی از کارکرد امضای کلاسی در مواقعی که متدهای کلاس دارای خروجی و ورودی‌های غیرجاوایی هستند را داشته‌باشد. با اضافه‌نمودن ویژگی‌های یکتا و تغییر امضای پژوهش ترکی، در کنار افزایش سرعت تشخیص برنامک‌های بازبسته‌بندی شده، خصوصا در بخش تشخیص کدهای کتابخانه‌ای، توانستیم دقت پژوهش جاری را به میزان خوبی بالا نگه‌داریم. با مقایسه‌ی برنامک‌های داخل مخزن داده‌ی آزمون، شامل ۱۱۸۱ جفت بازبسته‌بندی شده و غیر تقلبی، روش این پژوهش تنها در موارد مثبت‌غلط نسبت به پژوهش ترکی افت داشته‌است که موجب دقت پایین‌تری نسبت به آن شده‌است. این در حالی‌است که از نظر سرعت اجرا، بهبود ۶ برابری را در تشخیص مشاهده می‌کنیم.



شکل ۵ـ۶: مقایسه‌ی دقت و فراخوان پژوهش جاری و ترکی [۵]

۵ـ۳ـ۲ مقایسه‌ی دودویی همراه با طبقه‌بندی

در ابتدای بخش ۵ـ۳ـ۱ اشاره کردیم که پژوهش انجام‌شده توسط آقای ترکی را می‌توان از دوجنبه‌ی متفاوت بهبود بخشید. از منظر دوم، مقایسه‌ی تمامی برنامک‌های اندرویدی موجود در مخزن به ازای یک برنامک هدف ورودی، روشی زمان‌بر و بیهوده‌است چرا که بسیاری از برنامک‌های اندرویدی داخل مخزن هیچ شباهتی به برنامک ورودی ندارند. به همین جهت ما در این پژوهش از یک مرحله طبقه‌بندی به جهت کاهش فضای مقایسه‌ی دودویی پژوهش آقای ترکی در کنار امضای بهبود داده‌شده استفاده کرده‌ایم. پس از اعمال طبقه‌بند نزدیک‌ترین همسایه بر روی جفت اول و یا دوم از ۷۹۶ جفت برنامک بازبسته‌بندی شده، تنها ۱۹ جفت برنامک، خارج از بازه‌ی ۲۵۰ برنامک نزدیک به برنامک هدف قرار گرفته‌اند. به عبارت دیگر، پس از استخراج تمامی ۲۵۰ همسایه‌ی نزدیک به برنامک اندرویدی هدف، از میان جفت‌های بازبسته‌بندی شده، بیش از ٪۹۷ از آن‌ها، در بازه‌ی ۲۵۰ تایی از جفت تقلبی خود قرار گرفته‌اند. به علاوه، برای ٪۸۱ از جفت‌های بازبسته‌بندی شده، در هنگام اعمال طبقه‌بندی مبتنی بر نزدیک‌ترین همسایه، کمتر از ۲۰ برنامک با جفت تقلبی خود فاصله داشته‌اند. لازم به ذکر است، بازه‌ی ۲۰ تایی میان برنامک اصلی و برنامک تقلبی ممکن است شامل تعدادی از بسته‌های بازبسته‌بندی شده‌ی دیگری باشد که در لیست داده‌ی آزمون پژوهش که به صورت تصادفی انتخاب شده‌است نباشد. به علاوه، بررسی داده‌های اندروزو بر روی داده‌های ۲۲ میلیون برنامک اندرویدی، تمامی برنامک‌های بازبسته‌بندی شده را شامل نمی‌شود و تعدادی از برنامک‌ها در این لیست ۱۵۰۰۰ تایی آورده نشده‌اند. بنابراین، ارزیابی روش، به خصوص در مورد فاصله از جفت تقلبی، می‌تواند بسیار بهتر از ارزیابی فعلی باشد.

شکل ۵ـ۷: نمودار فاصله از جفت بازبسته‌بندی شده به ازای تمامی برنامک‌های بازبسته‌بندی شده

با استفاده از اجرای طبقه‌بندی بر روی برنامک‌های اندرویدی داخل مخزن، تعداد برنامک‌های مورد بررسی برای هر جفت از ۱۱۸۰ برنامک به ۲۵۰ برنامک نزدیک به خود کاهش پیدا کرد. ارزیابی روش ارائه‌شده مطابق نمودار ۵ـ۷ نشان می‌دهد که عملکرد طبقه‌بند نزدیک‌ترین همسایه، برای کاهش فضای مقایسه‌ی دودویی کاملا موثر می‌باشد چرا که فضای مقایسه‌ی دودویی را ۸۰ درصد کاهش داده‌است.

علاوه بر این ٪۹۷ از جفت‌های بازبسته‌بندی شده در فاصبله‌ی ۲۵۰ تایی از برنامک هدف قرار گرفته‌اند که موجب می‌شود طبقه‌بندی و اجرای مقایسه‌ی دودویی روی همسایه‌ها نزدیک، با توجه به بهبود سرعت امضا، منجر به افزایش سرعت مقایسه‌ی دودویی در کنار حفظ دقت پژوهش‌های اخیر گردد.
به‌جهت مقایسه‌ی زمانی پژوهش جاری و مقایسه‌ی دودویی معمول موجود در پژوهش آقای ترکی، فرض کنید که اجرا و مقایسه‌ی هر جفت برنامک مطابق با شکل ۵ـ۵ به ترتیب ۲۴ ثانیه و ۱۲۶ ثانیه زمان خواهد برد. در این صورت در مقایسه‌ی دودویی پژوهش صورت‌گرفته در روش آقای ترکی، برای یافتن جفت بازبسته‌بندی نیازمند صرف ۱۴۸۰۰۰ ثانیه زمان هستیم. این در حالی‌است که زمان میانگین مورد نیاز در پژوهش جاری، با توجه به میانگین فاصله‌ی برنامک از جفت بازبسته‌بندی شده که عدد ۱۶ می‌باشد، ۳۸۴ ثانیه‌است که اختلاف معناداری را به جهت مقایسه‌ی برنامک‌های اندرویدی ایجاد می‌کند.

۵ـ۴ تحلیل و جمع‌بندی عملکرد روش پیشنهادی

به طور کلی پژوهش جاری، پژوهش آقای وانگ [۵۳] و آقای ترکی [۵] را می‌توان در دسته‌ی روش‌های مبتنی بر تحلیل ایستا با استفاده از ویژگی‌های کدپایه به حساب آورد. هدف اصلی پژوهش وانگ، ارائه‌ی ابزاری به جهت تشخیص کدهای کتابخانه‌ای با استفاده از مقایسه‌ی دودویی ویژگی‌های کلاسی و یافتن کلاس‌های کتابخانه‌ای بوده‌است. استفاده از ایده‌ی ایجاد نگاشت میان کلاس‌های کتابخانه‌ای و کلاس‌های برنامک اولین بار در پژوهش آقای وانگ عنوان شده‌است و در ادامه توسط آقای ترکی با استفاده از فیلترهای طول امضا و ساختاری، بهبود پیدا کرده‌است. بنابراین پژوهش آقای وانگ را می‌توان سنگ‌بنای تشخیص کدهای کتابخانه‌ای در پژوهش ترکی و پژوهش جاری محسوب نمود. اما با توجه به تمرکز هر دو پژوهش، بهبود محسوسی در هیچ‌کدام از آن‌ها در قسمت تشخیص برنامک‌های بازبسته‌بندی شده مشاهده نشده است. هر دو پژوهش از همان ایده‌ی پرهزینه‌ی مطروحه در قسمت شباهت‌سنجی دودویی میان کلاس‌های برنامک و در نهایت شباهت‌سنجی کل برنامک‌های داخل مخزن با تمامی برنامک‌های دیگر استفاده کرده‌اند. علاوه بر این، از آن‌جایی که بسیاری از وابستگی‌های کلاسی نظیر وابستگی کلاس فرزند به کلاس پدر، کلاس‌های درونی و بیرونی و واسط‌های پیاده‌سازی شده، در پژوهش آقای ترکی به جهت افزایش دقت در تشخیص کدهای کتابخانه‌ای اضافه شده‌است بنابراین حجم امضای برنامک‌های اندرویدی در این پژوهش نسبت به پژوهش وانگ افزایش قابل توجهی داشته‌است زیرا امضای برنامک، جزئیات بیشتری از هر قسمت را با خود به همراه دارد. گرچه با استفاده از ایده‌ی فیلتر طول امضا در پژوهش آقای ترکی سعی شده تا قسمتی از این افزونگی‌داده حل‌شود اما هم‌چنان مقایسه‌ی امضای کلاس‌ها در قسمت نگاشت میان کلاس‌های کتابخانه‌ای بسیار پرهزینه است. ایده‌ی دیگری که در پژوهش آقای ترکی استفاده شده‌است، تطابق امضای متدها در گراف فراخوانی میان متدها است که در پژوهش آقای وانگ به آن توجهی نشده‌است. استفاده از

گراف فراخوان متد، موجب می‌شود تا امضای متدهایی که به صورت ایستا مبهم شده‌اند و داخل گراف فراخوان نیستند ساخته نشود.

تمرکز این پژوهش بر روی بهبود کارایی در قسمت تشخیص برنامک‌های اندرویدی بازبسته‌بندی شده بوده‌است. اگر چه بهبود امضای کلاس‌های برنامک و کاهش افزونگی‌های موجود در امضای کلاسی حاصل از پژوهش ترکی موجب شده‌است بخش تشخیص کدهای کتابخانه‌ای نیز بهبود پیدا کند اما حذف افزونگی‌های موجود در پژوهش ترکی با استفاده از جایگزینی ویژگی‌های طولانی با ویژگی‌های مختصر اما یکتا باعث‌شده است تا در قسمت مقایسه‌ی دودویی برنامک ها افزایش سرعت ۶ برابری داشته‌باشیم. علی‌رغم افزایش سرعت، همچنان ایده‌ی مقایسه‌ی دودویی یک برنامک با تمامی برنامک‌های داخل مخزن به جهت یافتن جفت تقلبی آن بیهوده و زمان‌بر به نظر می‌رسید. برای حل این مشکل، از یک ایده‌ی طبقه‌بندی استفاده شد. با استفاده از یک طبقه‌بند نزدیک‌ترین همسایه و ویژگی‌های مبتنی بر منابع برنامک‌های اندرویدی، مشکوک‌ترین برنامک‌ها را از داخل مخزن انتخاب و مقایسه‌ی آن‌ها را انجام می‌دهیم. طبقه‌بندی گرچه موجب کاهش یک درصدی دقت شده‌است، اما فضای مقایسه‌ی برنامک‌های اندرویدی را ۵ برابر کاهش داده که از نظر پژوهش مقرون به صرفه می‌باشد.

از نظر دسته‌بندی در روش‌های ایستا، پژوهش جاری را می‌توان ترکیبی از روش‌های مبتنی بر گراف و مبتنی بر منابع برنامک با استفاده از ویژگی‌های آن دانست. در این پژوهش برای تشکیل امضا از فراخوانی‌های موجود در گراف فراخوانی متد، واسط‌های برنامه‌نویسی و مجموعه‌ای از ویژگی‌های کدپایه استفاده شده‌است که تمامی آن‌ها در دسته‌ی تحلیل ایستا قرار می‌گیرند. از طرفی برای طبقه‌بندی از ویژگی‌های مبتنی بر منابع برنامک استفاده شده‌است چرا که منابع برنامک‌های اندرویدی تمایز درشت‌دانه میان برنامک‌ها برقرار می‌سازند.

فصل ۶

نتیجه‌گیری

به طور کلی روش‌های تشخیص برنامک‌های بازبسته‌بندی شده، برای تشخیص جفت بازبسته‌بندی شده باید برنامک‌های داخل مخزن را با برنامک هدف مقایسه کنند. استخراج ویژگی از برنامک‌های اندرویدی و مقایسه‌ی دوبه‌دوی آن‌ها با یکدیگر، ساختار اصلی اکثر پژوهش‌های موجود در این زمینه را تشکیل می‌دهد. تقریبا تمامی روش‌های مبتنی بر تحلیل ایستا برای مقایسه، سعی در استخراج مدلی از برنامک دارند که نشان‌دهنده‌ی مهم‌ترین ویژگی‌های ساختاری آن‌ها باشد. مقاومت روش ارائه‌شده در مقابل راهکارهای مبهم‌نگاری، به صورت مستقیم وابسته به مدلی‌است که روش پژوهش از آن برای نشان‌دادن برنامک استفاده می‌کند. بنابراین به هر میزان که ویژگی‌های منتخب یکتا باشند، مقاومت در مقابل مبهم‌نگاری و در نهایت دقت پژوهش نیز افزایش می‌یابد.

در کنار توجه به افزایش دقت در پژوهش‌های این حوزه، سرعت تشخیص نیز اهمیتی برابر دارد چرا که اگر ویژگی‌های منتخب و در نهایت مدل یکتای هر برنامک، ویژگی‌های متفاوتی را دربر داشته‌باشد و یا از روش‌های مبتنی بر گراف استفاده کند، آن‌گاه مقایسه‌ی مدل‌های موجود نیز سخت و زمان‌بر خواهد بود به طوری که در برخی از پژوهش‌های موجود در این حوزه، نظیر آن‌چه در پژوهش‌های مبتنی بر گراف دیده می‌شود، داده‌ی آزمون محدود به برنامک‌های کمی است که عملا مقیاس‌پذیری ارزیابی ارائه‌شده را نقض می‌کند و به کارگیری آن در یک ابزار کاربردی و تجاری را غیرممکن می‌نماید. از طرفی، پژوهش‌های موجود در زمینه‌ی تشخیص بازبسته‌بندی با استفاده از ویژگی‌های مبتنی بر منابع برنامک، گرچه سرعت بالایی را در تشخیص دارند اما از آن‌جایی که ایجاد مبهم‌نگاری در منابع، آسانتر از ویژگی‌های مبتنی بر کد برنامک است، روش‌های موجود در بهترین حالت می‌توانند راه‌حلی برای طبقه‌بندی درشت‌دانه را ارائه کنند و معمولا در تشخیص جفت بازبسته‌بندی شده، دقت خوبی را ارائه نمی‌دهند.

در این پژوهش ابتدا با استفاده از مولفه‌ی تشخیص کدهای کتابخانه‌ای و به کمک ایده‌ی استفاده از فیلترهای

ساختاری و طولی، موجود در پژوهش ترکی [۵]، فضای مقایسه‌ای به جهت تشخیص کدهای کتابخانه‌ای را کاهش دادیم و به کمک ماژول نگاشت، پس از استفاده از توابع چکیده‌سازی محلی $sdhash, ssdeep, tlsh$ و مقایسه‌ی کلاس‌های کتابخانه‌ای و کلاس‌های برنامک، لیستی از کلاس‌های کتابخانه‌ای کاندید را به ازای هر کلاس برنامک به دست آوردیم. در نهایت برای استخراج نگاشت میان کلاس‌های کتابخانه‌ای و کلاس‌های برنامک، مساله‌ی نگاشت میان کلاس‌های کتابخانه‌ای با استفاده از معکوس امتیاز تشابه را به مساله‌ی تخصیص در یک گراف دو بخشی تبدیل و در نهایت کلاس‌های کتابخانه‌ای برنامک را شناسایی می‌کنیم.

پس از استخراج کلاس‌های کتابخانه‌ای برنامک، حاصل از اعمال مولفه‌ی تشخیص کدهای کتابخانه‌ای، شروع به حذف آن‌ها کرده و کد اصلی برنامک را که توسط توسعه‌دهنده‌ی برنامک توسعه داده‌شده است را به دست می‌آوریم. در این قسمت ابتدا با استفاده از حذف افزونگی‌های پژوهش آقای ترکی و همچنین اضافه‌نمودن ویژگی‌های با طول کم‌تر، اندازه‌ی امضای نهایی برنامک‌های هدف را به طور قابل توجهی کاهش دادیم. به طوری که در قسمت مقایسه‌ی دودویی برنامک‌ها، سرعت تشخیص پژوهش جاری در مقایسه با پژوهش آقای ترکی ۶ برابر و فراخوان پژوهش تنها ۱ درصد افت را تجربه کرده‌است.

از جهت دیگر، به جهت کاهش فضای مقایسه‌ی برنامک‌های اندرویدی، ایده‌ی استفاده از یک طبقه‌بندی مبتنی بر نزدیک‌ترین همسایه با استفاده از ویژگی‌های مبتنی بر منابع مطرح شده‌است. استفاده از ویژگی‌های مبتنی بر منابع در کنار مقایسه‌ی دودویی با استفاده از ویژگی‌های مبتنی بر کد برنامک، منجر به مقاومت مناسب این روش در مقابل روش‌های مبهم‌نگاری شده است به طوری که از میان ۷۹۶ جفت بازبسته‌بندی شده تنها ۱۹ جفت در بازه‌ی ۲۵۰ تایی برنامک‌های اندرویدی هدف قرار نگرفته‌اند که منجر به کاهش حدود ۵ برابری فضای مقایسه شده‌است.

به طور کلی ، با مقایسه‌ی روش این پژوهش و پژوهش آقای ترکی، در قسمت مقایسه‌ی دودویی برنامک‌ها، با حفظ میزان دقت ٪۹۸ و کاهش ۱ درصدی فراخوان از ٪۹۸ به ٪۹۷ توانستیم سرعت روش این پژوهش را در حالت مقایسه‌ی دودویی ۶ برابر افزایش دهیم. علاوه بر این استفاده از طبقه‌بند مبتنی بر نزدیک‌ترین همسایه، دقتی معادل ٪۹۷ را در این پژوهش به همراه داشته‌است و در کنار آن، منجر به کاهش ۵ برابری فضای مقایسه‌ی برنامک‌های اندرویدی شده‌است.

در انتها روش پیشنهادی در این پژوهش را می‌توان با استفاده از روش‌های متفاوت که در ادامه به آن‌ها اشاره‌شده است بهبود بخشید:

- یکی از ضعف‌های این پژوهش و اصولا تمامی پژوهش‌های مبتنی بر تحلیل ایستا با استفاده از گراف جریان، اتکای ویژگی‌های آماری هر برنامک به گراف فراخوانی میان متدهای آن می‌باشد. پژوهش‌های موجود در مقابل اکثر روش‌های مبهم‌نگاری خصوصا روش‌های مبهم‌نگاری که به صورت رایگان

در اختیار بدافزارنویسان هستند مقاوم هستند اما در صورتی که گراف فراخوانی برنامک دچار ابهام شود، تشخیص برنامک با استفاده از گراف فراخوانی باید با استفاده از روش‌های دیگر نظیر تحلیل پویای رفتار برنامک و بهبود سرعت آن، صورت بگیرد. از طرفی، تعدد برنامک‌های بازبسته‌بندی شده که به صورت رایگان مبهم شده‌اند نیز آنقدر زیاد است که مقابله‌ی با آن‌ها را حیاتی کرده‌است چرا که برنامک‌های تجاری خود به صورت ذاتی از روش‌های پیشگیری از بازبسته‌بندی استفاده می‌کنند.

- داده‌ی آزمون اکثر پژوهش‌های مطروحه در دسته‌ی تشخیص برنامک‌های بازبسته‌بندی شده عمومی نیستند و در بسیاری از آن‌ها جتی از ذکر جزئیات نحوه‌ی مبهم‌نگاری و یا فروض اولیه‌ی تعریف بازبسته‌بندی نیز خودداری شده‌است. به علاوه برنامک‌های تجاری نیز پیچیدگی‌های خاص خود را دارند که تشخیص بازبسته‌بندی در این دسته از برنامک‌ها را با مشکل مواجه می‌کند. ساخت یک داده‌ی آزمون مناسب شامل برنامک‌های تجاری و رایگان که بازبسته‌بندی شده‌اند، در کنار شفافیت در مورد چگونگی مبهم‌نگاری در آن‌ها می‌تواند به عنوان یک ایده‌ی پژوهشی مناسب در این زمینه مطرح شود.
- طبقه‌بندی در این پژوهش با اتکا به منابع ایستا در پوشه‌ی مربوط به واسط کاربری برنامک صورت می‌گیرد. تحلیل فراخوانی هر کدام از انواع منابع در کد برنامک‌های اندرویدی و در نهایت اجرای طبقه‌بندی روی ویژگی‌های مبتنی بر این دسته، می‌تواند به عنوان یک ایده برای افزایش دقت مرحله‌ی طبقه‌بندی استفاده شود.

مراجع

[1] Y. Shao, X. Luo, C. Qian, P. Zhu, and L. Zhang. Towards a Scalable Resource-Driven Approach for Detecting Repackaged Android Applications. In *proceedings of 30th Annual Computer Security Applications Conference*, ACSAC '14, pages 56–65, New York, NY, USA, 2014. ACM.

[2] M. S. Bhatt, H. Patel, and S. Kariya. A Survey Permission Based Mobile Malware Detection. *International journal Computer Technology and Applications*, 6(5):852–856, 2015.

[3] L. Adhianto, S. Banerjee, M. Fagan, M. Krentel, G. Marin, J. Mellor-Crummey, and N. R. Tallent. Hpctoolkit: Tools for Performance Analysis of Optimized Parallel Programs. *Concurrency and Computation: Practice and Experience*, 22(6):685–701, 2010.

[4] N. T. Cam, N. H. Khoa, T. T. An, N. P. Bach, and V.-H. Pham. Detect Repackaged Android Applications by Using Representative Graph. In *proceedings of 8th NAFOSTED Conference on Information and Computer Science (NICS)*, pages 102–106, 2021.

[5] M. Torki. Detecting Repackaged Android Applications . Master's thesis, Sharif University Of Technology, Feb. 2018.

[6] Global mobile OS market share 2022 Statista statista.com. https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/#:~:text=Android%20maintained%20its%20position%20as,the%20mobile%20operating%20system%20market. [Accessed 02-Feb-2023].

[7] Play Protect | Google Developers developers.google.com. https://developers.google.com/android/play-protect. [Accessed 02-Feb-2023].

[8] Decompile and modify an Android application | cylab.be — cylab.be. https://cylab.be/blog/69/decompile-and-modify-an-android-application. [Accessed 02-Feb-2023].

[9] A. Dizdar. OWASP Mobile Top 10 Vulnerabilities and How to Prevent Them — brightsec.com. https://brightsec.com/blog/owasp-mobile-top-10/. [Accessed 02-Feb-2023].

[10] D. J. Wu, C. H. Mao, T. E. Wei, H. M. Lee, and K. P. Wu. Droidmat: Android Malware Detection Through Manifest and Api Calls Tracing. In *Proceedings of 7th Asia Joint Conference on Information Security, AsiaJCIS 2012*, pages 62–69, 2012.

[11] K. Khanmohammadi, N. Ebrahimi, A. Hamou-Lhadj, and R. Khoury. Empirical Study of Android Repackaged Applications. *Empirical Software Engineering*, 24(6):3587–3629, 2019.

[12] T. Vidas and N. Christin. Sweetening Android Lemon Markets: Measuring And Combating Malware in Application Marketplaces. In *Proceedings of the 3rd ACM Conference on Data and Application Security and Privacy*, volume 2011, pages 197–207, 2013.

[13] P. Maniriho, A. N. Mahmood, and M. J. M. Chowdhury. A Study on Malicious Software Behaviour Analysis and Detection Techniques: Taxonomy, Current Trends and Challenges. *Future Generation Computer Systems*, 130:1–18, 2022.

[14] Z. Ma, H. Wang, Y. Guo, and X. Chen. LibRadar: Fast and Accurate Detection of Third-Party Libraries in Android Apps. In *proceedings of IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*, pages 653–656, 2016.

[15] S. Dong, M. Li, W. Diao, X. Liu, J. Liu, Z. Li, F. Xu, K. Chen, X. F. Wang, and K. Zhang. *Understanding Android Obfuscation Techniques: a Large-scale Investigation in the Wild*, volume 254. Springer International Publishing, 2018.

[16] V. Rastogi, Y. Chen, and X. Jiang. Droidchameleon: Evaluating Android Anti-malware Against Transformation Attacks. In *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*, pages 329–334, 2013.

[17] Trail: the reflection api the javax; tutorials — docs.oracle.com. https://docs.oracle.com/javase/tutorial/reflect/index.html. [Accessed 02-Feb-2023].

[18] X. Zhang, F. Breitinger, E. Luechinger, and S. O'Shaughnessy. Android Application Forensics: a Survey of Obfuscation, Obfuscation Detection and Deobfuscation Techniques and Their Impact on Investigations. *Forensic Science International: Digital Investigation*, 39:301285, 2021.

[19] ProGuard Manual: Home | Guardsquare — guardsquare.com. https://www.guardsquare.com/manual/home. [Accessed 02-Feb-2023].

[20] Allatori Java Obfuscator — codedemons.net. http://www.codedemons.net/allatori.html. [Accessed 02-Feb-2023].

[21] Y. Wang. Obfuscation-Resilient Code Detection Analyses for Android Apps. 2018.

[22] L. Ardito, R. Coppola, S. Leonardi, M. Morisio, and U. Buy. Automated Test Selection for Android Apps Based on APK and Activity Classification. *IEEE Access*, 8:187648–187670, 2020.

[23] L. Li, T. F. Bissyandé, J. Klein, and Y. Le Traon. An Investigation into the Use of Common Libraries in Android Apps. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, volume 1, pages 403–414, 2016.

[24] N. Karankar, P. Shukla, and N. Agrawal. Comparative Study of Various Machine Learning Classifiers on Medical Data. pages 267–271, 2017.

[25] Y. Shao, X. Luo, C. Qian, P. Zhu, and L. Zhang. Towards a Scalable Resource-driven Approach for Detecting Repackaged Android Applications. *ACM*, 2014-Decem(December):56–65, 2014.

[26] F. Zhang, H. Huang, S. Zhu, D. Wu, and P. Liu. ViewDroid: Towards Obfuscation-Resilient Mobile Application Repackaging Detection. pages 25–36, 2014.

[27] J. Crussell, C. Gibler, and H. Chen. Attack of the Clones: Detecting Cloned Applications on Android Markets. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7459 LNCS:37–54, 2012.

[28] H. Gonzalez, N. Stakhanova, and A. A. Ghorbani. Droidkin: Lightweight Detection of Android Apps Similarity. *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST*, 152(January 2015):436–453, 2015.

[29] X. Chen, C. Li, D. Wang, S. Wen, J. Zhang, S. Nepal, Y. Xiang, and K. Ren. Android HIV: A Study of Repackaging Malware for Evading Machine-Learning Detection. *IEEE Transactions on Information Forensics and Security*, 15(8):987–1001, 2020.

[30] A. Salem. Stimulation and Detection of Android Repackaged Malware with Active Learning. 2015.

[31] T. Nguyen, J. T. McDonald, W. B. Glisson, and T. R. Andel. Detecting Repackaged Android Applications Using Perceptual Hashing. *Proceedings of the Annual Hawaii International Conference on System Sciences*, 2020-January:6641–6650, 2020.

[32] F. Alswaina and K. Elleithy. Android Malware Family Classification and Analysis: Current Status and Future Directions. *Electronics (Switzerland)*, 9(6):1–20, 2020.

[33] F. Akbar, M. Hussain, R. Mumtaz, Q. Riaz, A. W. A. Wahab, and K.-H. Jung. Permissions-Based Detection of Android Malware Using Machine Learning. *Symmetry*, 14(4), 2022.

[34] X. Chen, C. Li, D. Wang, S. Wen, J. Zhang, S. Nepal, Y. Xiang, and K. Ren. Android HIV: A Study of Repackaging Malware for Evading Machine-Learning Detection. *IEEE Transactions on Information Forensics and Security*, 15:987–1001, 2020.

[35] Q. Zhang, X. Zhang, Z. Yang, and Z. Qin. An Efficient Method of Detecting Repackaged Android Applications. pages 056 (4 .)–056 (4 .), 01 2014.

[36] X. Sun, Y. Zhongyang, Z. Xin, B. Mao, L. Xie, X. Sun, Y. Zhongyang, Z. Xin, B. Mao, L. Xie, D. Code, X. Sun, Y. Zhongyang, Z. Xin, B. Mao, and L. Xie. Detecting Code Reuse in Android Applications Using Component-based Control Flow Graph to Cite This Version : Hal Id : Hal-01370361 Detecting Code Reuse in Android Applications Using Component-based Control Flow Graph. pages 0–14, 2016.

[37] W. Zhou, Y. Zhou, X. Jiang, and P. Ning. Detecting Repackaged Smartphone Applications in Third-Party Android Marketplaces. In *Proceedings of the Second ACM Conference on Data and Application Security and Privacy*, CODASPY '12, pages 317–326, New York, NY, USA, 2012. ACM.

[38] A. Desnos. Android: Static Analysis Using Similarity Distance. In *proceedings of 45th Hawaii International Conference on System Sciences*, pages 5394–5403, 2012.

[39] B. B. Rad and M. Masrom. Metamorphic Virus Detection in Portable Executables Using Opcodes Statistical Feature. *International Journal on Advanced Science, Engineering and Information Technology*, 1(4):403, 2011.

[40] B. B. Rad, M. Masrom, and S. Ibrahim. Opcodes Histogram for Classifying Metamorphic Portable Executables Malware. In *proceedings of International Conference on E-Learning and E-Technologies in Education (ICEEE)*, pages 209–213, 2012.

[41] Q. Jerome, K. Allix, R. State, and T. Engel. Using Opcode-sequences to Detect Malicious Android Applications. In *proceedings of IEEE International Conference on Communications (ICC)*, pages 914–919, 2014.

[42] C. Liangboonprakong and O. Sornil. Classification of Malware Families Based on N-grams Sequential Pattern Features. In *proceedings of IEEE 8th Conference on Industrial Electronics and Applications (ICIEA)*, pages 777–782, 2013.

[43] Y. D. Lin, Y. C. Lai, C. H. Chen, and H. C. Tsai. Identifying Android Malicious Repackaged Applications by Thread-grained System Call Sequences. *Computers and Security*, 39(PART B):340–350, 2013.

[44] P. Faruki, V. Laxmi, V. Ganmoor, M. Gaur, and A. Bharmal. DroidOLytics: Robust Feature Signature for Repackaged Android Apps on Official and Third Party Android Markets. In *proceedings of 2nd International Conference on Advanced Computing, Networking and Security*, pages 247–252, 2013.

[45] J. Ko, H. Shim, D. Kim, Y.-S. Jeong, S.-j. Cho, M. Park, S. Han, and S. B. Kim. Measuring Similarity of Android Applications via Reversing and K-Gram Birthmarking. In *Proceedings of Research in Adaptive and Convergent Systems*, RACS '13, pages 336–341, New York, NY, USA, 2013. ACM.

[46] S. Kishore, R. Kumar, and S. Rajan. *Towards Accuracy in Similarity Analysis of Android Applications*, volume 11281 LNCS. Springer International Publishing, 2018.

[47] R. Potharaju, A. Newell, C. Nita-Rotaru, and X. Zhang. Plagiarizing Smartphone Applications: Attack Strategies and Defense Techniques. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7159 LNCS:106–120, 2012.

[48] J. Crussell, C. Gibler, and H. Chen. Scalable Semantics-Based Detection of Similar Android Applications. *Esorics*, pages 182–199, 2013.

[49] W. Hu, J. Tao, X. Ma, W. Zhou, S. Zhao, and T. Han. Migdroid: Detecting App-repackaging Android Malware via Method Invocation Graph. In *proceedings of 23rd International Conference on Computer Communication and Networks (ICCCN)*, pages 1–7, 2014.

[50] W. Zhou, Y. Zhou, M. Grace, X. Jiang, and S. Zou. Fast, Scalable Detection of "Piggybacked" Mobile Applications. In *Proceedings of the 3rd ACM Conference on Data and Application Security and Privacy*, pages 185–195, 2013.

[51] K. Chen, P. Liu, and Y. Zhang. Achieving Accuracy and Scalability Simultaneously in Detecting Application Clones on Android Markets. In *Proceedings of International Conference on Software Engineering*, number 1, pages 175–186, 2014.

[52] J. Zheng, K. Gong, S. Wang, Y. Wang, and M. Lei. Repackaged Apps Detection Based on Similarity Evaluation. In *Proceedings of 8th International Conference on Wireless Communications & Signal Processing (WCSP)*, pages 1–5, 2016.

[53] Y. Wang. *Obfuscation-Resilient Code Detection Analyses for Android Apps*. PhD thesis, Ohio State University, Aug. 2018.

[54] X. Wu, D. Zhang, X. Su, and W. Li. Detect Repackaged Android Application Based on Http Traffic Similarity. *Sec. and Commun. Netw.*, 8(13):2257–2266, sep 2015.

[55] M. Alshehri. App-nts: a Network Traffic Similarity-based Framework for Repacked Android Apps Detection. *Journal of Ambient Intelligence and Humanized Computing*, 13(3):1537–1546, 2022.

[56] G. He, B. Xu, L. Zhang, and H. Zhu. On-Device Detection of Repackaged Android Malware via Traffic Clustering. *Security and Communication Networks*, 2020:8630748, May 2020.

[57] A. Rodriguez and A. Laio. Clustering by Fast Search and Find of Density Peaks. *Science*, 344(6191):1492–1496, 2014.

[58] J. Malik and R. Kaushal. Credroid: Android Malware Detection by Network Traffic Analysis. In *Proceedings of 1st ACM Workshop on Privacy-Aware Mobile Computing*, PAMCO '16, pages 28–36, New York, NY, USA, 2016. ACM.

[59] D. Iland and A. Pucher. Detecting Android Malware on Network Level. 2011.

[60] S. Kandukuru and R. M. Sharma. Android Malicious Application Detection Using Permission Vector and Network Traffic Analysis. In *Proceedings of 2nd International Conference for Convergence in Technology (I2CT)*, pages 1126–1132, 2017.

[61] M. Lin, D. Zhang, X. Su, and T. Yu. Effective and Scalable Repackaged Application Detection Based on User Interface. In *Proceedings of IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 1–6, 2017.

[62] S. Yue, Q. Sun, J. Ma, X. Tao, C. Xu, and J. Lu. RegionDroid: A Tool for Detecting Android Application Repackaging Based on Runtime UI Region Features. In *Proceedings of IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 323–333, 2018.

[63] Y. Hu, G. Xu, B. Zhang, K. Lai, G. Xu, and M. Zhang. Robust App Clone Detection Based on Similarity of UI Structure. *IEEE Access*, 8:77142–77155, 2020.

[64] S. Li. Juxtapp and dstruct: Detection of similarity among android applications. Master's thesis, EECS Department, University of California, Berkeley, May 2012.

[65] J. Guo, D. Liu, R. Zhao, and Z. Li. WLTDroid: Repackaging Detection Approach for Android Applications. In G. Wang, X. Lin, J. Hendler, W. Song, Z. Xu, and G. Liu, editors, *proceedings of Web Information Systems and Applications: 17th International Conference, WISA 2020, Guangzhou, China, September 23–25, 2020, Proceedings*, pages 579–591, Cham, 2020. Springer International Publishing.

[66] F. Lyu, Y. Lin, J. Yang, and J. Zhou. SUIDroid: An Efficient Hardening-Resilient Approach to Android App Clone Detection. In *proceedings of IEEE Trustcom/BigDataSE/ISPA*, pages 511–518, 2016.

[67] X. Liu, Z. Yu, Z. Song, L. Chen, and Z. Qin. MDSDroid: A Multi-level Detection System for Android Repackaged Applications. In *proceedings of IEEE 6th International Conference on Signal and Image Processing (ICSIP)*, pages 1128–1133, 2021.

[68] S. Yue, W. Feng, J. Ma, Y. Jiang, X. Tao, C. Xu, and J. Lu. RepDroid: An Automated Tool for Android Application Repackaging Detection. In *Proceedings of IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*, pages 132–142, 2017.

[69] J. Ma, Q.-W. Sun, C. Xu, and X.-P. Tao. GridDroid—An Effective and Efficient Approach for Android Repackaging Detection Based on Runtime Graphical User Interface. *Journal of Computer Science and Technology*, 37(1):147–181, Feb 2022.

[70] V. Costamagna, C. Zheng, and H. Huang. Identifying and Evading Android Sandbox Through Usage-Profile Based Fingerprints. In *Proceedings of the First Workshop on Radical and Experiential Security*, RESEC '18, pages 17–23, New York, NY, USA, 2018. ACM.

[71] Y. Zhang, K. Huang, Y. Liu, K. Chen, L. Huang, and Y. Lian. Timing-Based Clone Detection on Android Markets. 12 2015.

[72] T. Nguyen, J. T. McDonald, W. B. Glisson, and T. R. Andel. Detecting Repackaged Android Applications Using Perceptual Hashing. In *Proceedings of 53rd Hawaii International Conference on System Sciences*, 2020.

[73] C. Soh, H. B. Kuan Tan, Y. L. Arnatovich, and L. Wang. Detecting Clones in Android Applications through Analyzing User Interfaces. In *2015 IEEE 23rd International Conference on Program Comprehension*, pages 163–173, 2015.

[74] T. Bläsing, L. Batyuk, A.-D. Schmidt, S. A. Camtepe, and S. Albayrak. An Android Application Sandbox System for Suspicious Software Detection. In *Proceedings of 5th International Conference on Malicious and Unwanted Software*, pages 55–62, 2010.

[75] D. Kim, A. Gokhale, V. Ganapathy, and A. Srivastava. Detecting Plagiarized Mobile Apps Using Api Birthmarks. *Automated Software Engineering*, 23(4):591–618, 2016.

[76] Q. Guan, H. Huang, W. Luo, and S. Zhu. Semantics-Based Repackaging Detection for Mobile Apps. *Engineering Secure Software and Systems*, pages 89–105, 2016.

[77] H. Wang, Y. Guo, Z. Ma, and X. Chen. WuKong: A Scalable and Accurate Two-Phase Approach to Android App Clone Detection. In *Proceedings of International Symposium on Software Testing and Analysis*, ISSTA 2015, pages 71–82, New York, NY, USA, 2015. ACM.

[78] O. S. J. Nisha and S. M. S. Bhanu. Detection of repackaged Android applications based on Apps Permissions. In *Proceedings of 4th International Conference on Recent Advances in Information Technology (RAIT)*, pages 1–8, 2018.

[79] W. Zhou, X. Zhang, and X. Jiang. AppInk: Watermarking Android Apps for Repackaging Deterrence. In *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*, ASIA CCS '13, pages 1–12, New York, NY, USA, 2013. ACM.

[80] Y. Zhang and K. Chen. AppMark: A Picture-Based Watermark for Android Apps. In *Proceedings of Eighth International Conference on Software Security and Reliability (SERE)*, pages 58–67, 2014.

[81] C. Ren, K. Chen, and P. Liu. Droidmarking: Resilient Software Watermarking for Impeding Android Application Repackaging. In *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, ASE '14, pages 635–646, New York, NY, USA, 2014. ACM.

[82] X. Sun, J. Han, H. Dai, and Q. Li. An Active Android Application Repacking Detection Approach. In *Proceedings of 10th International Conference on Communication Software and Networks (ICCSN)*, pages 493–496, 2018.

[83] L. Luo, Y. Fu, D. Wu, S. Zhu, and P. Liu. Repackage-Proofing Android Apps. In *Proceedings of 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 550–561, 2016.

[84] Q. Zeng, L. Luo, Z. Qian, X. Du, and Z. Li. Resilient Decentralized Android Application Repackaging Detection Using Logic Bombs. In *Proceedings of International Symposium on Code Generation and Optimization*, CGO 2018, pages 50–61, New York, NY, USA, 2018. ACM.

[85] S. Tanner, I. Vogels, and R. Wattenhofer. Protecting Android Apps From Repackaging Using Native Code. In A. Benzekri, M. Barbeau, G. Gong, R. Laborde, and J. Garcia-Alfaro, editors, *Proceedings of Foundations and Practice of Security*, pages 189–204, Cham, 2020. Springer International Publishing.

[86] D. Wermke, N. Huaman, Y. Acar, B. Reaves, P. Traynor, and S. Fahl. A Large Scale Investigation of Obfuscation Use in Google Play. In *Proceedings of the 34th Annual Computer Security Applications Conference*, ACSAC '18, pages 222–235, New York, NY, USA, 2018. ACM.

[87] Virus Bulletin :: Obfuscation in Android malware, and how to fight back — virusbulletin.com. https://www.virusbulletin.com/virusbulletin/2014/07/obfuscation-android-malware-and-how-fight-back. [Accessed 12-Feb-2023].

[88] Y. Wang and A. Rountev. Who Changed You? Obfuscator Identification for Android. In *2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, pages 154–164, 2017.

[89] S. Alam, Z. Qu, R. Riley, Y. Chen, and V. Rastogi. Droidnative: Automating and Optimizing Detection of Android Native Code Malware Variants. *Computers and Security*, 65:230–246, 2017.

[90] A. Bacci, A. Bartoli, F. Martinelli, E. Medvet, and F. Mercaldo. Detection of Obfuscation Techniques in Android Applications. In *Proceedings of the 13th International Conference on Availability, Reliability and Security*, ARES 2018, New York, NY, USA, 2018. ACM.

[91] M. D. Preda and F. Maggi. Testing Android Malware Detectors Against Code Obfuscation: a Systematization of Knowledge and Unified Methodology. *Journal of Computer Virology and Hacking Techniques*, 13:209–232, 2017.

[92] Z. Li, J. Sun, Q. Yan, W. Srisa-an, and Y. Tsutano. Obfusifier: Obfuscation-Resistant Android Malware Detection System. In S. Chen, K.-K. R. Choo, X. Fu, W. Lou, and A. Mohaisen, editors, *Proceedings of Security and Privacy in Communication Networks*, pages 214–234, Cham, 2019. Springer International Publishing.

[93] L. Zhang, H. Meng, and V. L. L. Thing. Progressive Control Flow Obfuscation for Android Applications. In *Proceedings of IEEE Region 10 Conference*, pages 1075–1079, 2018.

[94] V. Rastogi, Y. Chen, and X. Jiang. Droidchameleon: Evaluating Android Anti-malware Against Transformation Attacks. In *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*, ASIA CCS '13, pages 329–334, New York, NY, USA, 2013. ACM.

[95] D. Maiorca, D. Ariu, I. Corona, M. Aresu, and G. Giacinto. Stealth Attacks: an Extended Insight Into the Obfuscation Effects on Android Malware. *Computers & Security*, 51:16–31, 2015.

[96] W. Hou, D. Li, C. Xu, H. Zhang, and T. Li. An Advanced k Nearest Neighbor Classification Algorithm Based on KD-tree. In *Proceedings of IEEE International Conference of Safety Produce Informatization (IICSPI)*, pages 902–905, 2018.

[97] K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon. AndroZoo: Collecting Millions of Android Apps for the Research Community. In *Proceedings of the 13th International Conference on Mining Software Repositories*, MSR '16, pages 468–471, New York, NY, USA, 2016. ACM.

واژه‌نامه

الف

اندروید ..... Android
استاتیستا ..... Statista
اندروزو ..... Androzoo
ایستا ..... Static
امضا ..... signature
آلاتوری ..... Allatori
اسماردک ..... Smardec
الگوریتم ..... algorithm
استخراج ویژگی ..... Feature Extracting
آپکد ..... Opcode
آنتروپی ..... Entropy
ابر ..... Cloud
آماری ..... Statistical
فیلتر تصمیم ..... Intent Filter
اسکی ..... Ascii
انتزاعی ..... Abstract
ارث‌بری کلاس ..... Class Inheritance

ب

برخط ..... online
بدافزار ..... Malware
بازبسته‌بندی نرم‌افزار ..... software Repackaging
برنامک اصلی ..... Original Application
بایت‌کد دالویک ..... Dalvik Byte Code
بومی ..... Native
بازتاب ..... Reflect
بارگیری ..... Load
بارگذاری پویای کلاس ..... Dynamic Class Loading
بسته ..... Module
بردار ویژگی ..... Feature vector
بلوم فیلتر ..... bloom Filter
برچسب‌زنی ..... Labeling
بمب منطقی ..... Logic bomb

پ

پلتفرم ..... platform
پلی پروتکت ..... Play Protect
پویا ..... Dynamic
پروگارد ..... progaurd
پیکربندی ..... Configuration
پیش‌پردازش ..... Preprocess
پنجره‌ی لغزان ..... Sliding window
پَرفایل استفاده ..... Usage Profile
پردازنده ..... CPU
پیشوند ..... Name Space
پرش قطعی ..... Unconditional Jump
بدنه‌ی متد ..... Method Body

بومی ..... Native

ت

توسعه ..... Development
توسعه‌دهندگان ..... Developers
تشخیص ..... Detection
تقسیم‌بندی ..... partition
توری ..... mesh
توزیع‌شده ..... distributed
تجاری ..... Commercial
ترافیک شبکه ..... Network Traffic
تطبیق جریان ترافیک ... Traffic Stream Matching
تطبیق هانگرین ..... Hungarian Matching
تطابق رشته ..... String Matching
تصاویر ضبط شده ..... Screen Shot
تغییردهنده‌ی متد ..... Method Modifier

ث

ثبات داده ..... Data Register

ج

جاوا ..... Java
جمع‌آزما ..... Checksum
جعبه‌شن ..... Sand Box
جریان دسترسی ..... Request Flow

چ

چکیده ..... Hash
چکیده‌سازی ادراکی ..... Perception Hashing
چکیده‌سازی محلی ..... Locality Sensitive Hash

ح

حساس ..... Sensitive
حریصانه ..... greedy
حق‌تکثیر ..... Copy right
حد آستانه ..... Threshold

خ

خوشه ..... cluster
خطی ..... linear

د

دانلود ..... Download
دی‌کامپایل ..... Decompile
داده‌ی پرت ..... outlier data
دوبرابرسازی ..... doubling
دودویی ..... binary
دکس گارد ..... Dex Gaurd
دودویی ..... Binary
درویدمس ..... DroidMoss
درخت نحو انتزاع ..... Abstract syntax Tree
درخت VP ..... Vontage-Point tree
درخواست نام دامنه ..... Domain Name Request
درخت تصمیم ..... Decision Tree
درخت طرح‌بندی کامل ..... Total Layout Tree
درشت‌دانه ..... Coars Grain
داده‌ساختار ..... Data Structure
دقت ..... Precision

ر

رابط کاربری ..... User Interface
رشته ..... String

رمزنگاری ..... Encryption
رمزگشایی ..... Decryption
رمز سزار ..... Caesar Cipher
ریزدانه ..... Fine Grain

ز

زمان اجرا ..... Execution Time

س

سیستم‌عامل ..... Operation System
سطح بالا ..... High Level
سطح معناشناسی ..... Semantic level
ساختار کنترلی ..... Control's Statement
ساختاری ..... Structural
سربرگ ..... Header
سلسله مراتب فایل ..... File Structure
ساختار شرطی ..... Conditional statement
سازنده ..... constructor

ش

شباهت‌سنجی ..... Similarity
شناسه‌ها ..... Identifier
شناسه‌ی منبع ..... Resource Id
شبیه‌ساز ..... Simulator
شاخص تصمیم ..... Decision Index
شما ..... schema

ص

صحت‌سنجی ..... Integrity

ط

طبقه‌بندی ..... Classification
طبقه‌بند ..... Classifier
طبقه‌بندی طیفی ..... Spectral Classification
طرح‌بندی ..... Layout

ع

عملوند ..... Operand
عبارت منظم ..... Regular Expression
عمق کلاس ..... Class's Depth

غ

غلبه ..... dominate

ف

فایل دکس ..... Dex file
فراخواننده ..... Caller
فایل اشکال‌زدایی ..... Debug File
فاصله ویرایشی ..... Edit Distance
فراخوانی سیستمی ..... System Call
فاصله‌ی فشرده‌سازی ..... Compression Distance
فراداده ..... Metadata
فاصله اقلیدوسی ..... Euclidian Distance
فعالیت ..... Activity
فراخوان ..... Recall

ق

قطعی ..... deterministic

ک

کامپایل ..... Compile
کدهای تبلیغاتی ..... Ad Code

کدپایه ..... Code Base
کتابخانه ..... Library
کلیدعمومی ..... Public Key
کد مرده ..... Dead Code
کارخواه‌ـ کارگزار ..... client-Server
کنترل‌گر رویداد ..... Event Handler
کلاس پدر ..... Parent Class
کاندید ..... Candidate
کلاس منطبق ..... Matched Class

گ

گوگل ..... Google
گراف فراخوانی ..... Call graph
گراف جریان ..... Flow Graph
گارداسکوار ..... Guard Square
گراف هم‌ریخت ..... Graph Isomorphism
گراف فراخوانی متد ... Method Invocation Graph
گراف فعالیت ..... Activity Graph
گراف انتزاعی طرح‌بندی . Abstract Layout Graph
گراف انتقالی انتقالی . Abstract Transition Graph
گزارش ..... Log
گراف فراخوانی کلاس ..... Class Call graph

ل

لیست سفید ..... White List

م

مهندسی معکوس ..... Reverse Engineering
مهاجم ..... Attacker
ماشین مجازی جاوا ..... Java Virtual Machine
مبهم‌نگاری ..... Obfuscation
مجموعه‌داده ..... Data Set
منابع ..... Resource
منطق ..... Logic
منبع پایه ..... Resource Base
ماون ..... Maven
مقایسه‌ی دودویی ..... Pairwise comparison
متد ..... Method
مخزن ماون ..... Maven Repository
معناشناسی ..... Semantic
متغیر ..... Variable
مبهم‌نگار ..... Obfuscator
مؤلفه ..... Component
مرتب‌سازی ..... Sorting
مسأله‌ی تصمیم‌گیری ..... Decision Problem
منیفست ..... Manifest
مجازی ..... Virtual
مجوز ..... Permissions
مرجع ..... Source
محتوای بسته ..... Packet Content
معناشناسی ..... Semantic
ماژول ..... Module
میانه ..... Median

ن

نزدیک‌ترین همسایه ..... Nearest Neighbor
نحو ..... syntax
نوپ ..... Nop
نیم‌فضا ..... half space
نگاشت ..... Mapping
نخ ..... thread
نقطه‌ی دسترسی .....
نمونه‌ی کلاسی ..... Class Instance

هرزنامه ..... Spam

هسته ..... Kernel

ی

یادگیری ماشین ..... Machine Learning

یافتن بزرگ‌ترین زیردنباله‌ی مشترک ..... Longest Common Subsequence

یکتا ..... Unique

و

ویروس توتال ..... Virus Total

وابستگی داده‌ای ..... Data Dependency

وابستگی کنترلی ..... Control Dependency

واسط پیاده‌سازی شده .... Implemented Interface

ه

Abstract

Attacks on Android devices often take the form of repackaging. Attackers change a well-known app that has been downloaded from the app store, reverse engineer it, add some malicious payloads, and then upload the modified app to the app store. Because it is difficult for users to distinguish between the changed app and the original app, users can be easily duped. The malicious code inside the modified apps can launch attacks after they are installed, typically in the background. There are so many repackage detection method proposed during last years of researches in this area. developing an approach to detect android repackaged application should contains two main goal, speed and accuracy of method. if an approach have a good execution detection time but detect in less accurate, there is still some pair that doesn't detect. on the other hand, accurate methods with less speed are useless because we can not use it in a real time system. two categories of techniques are currently used to combat the repackaging of Android applications. some of methods are based on repackaged detection before the attacker's modification. On the other hand, some methods proposed to prevent repackaging attack by watermarking or application obfuscation methods. Repackaged detection method divided in to two main category, static and dynamic analysis. Static method, extract the application feature that are resistant against obfuscation methods then, after third-parties removed, comparison between application done by features compassion. the method's speed and accuracy depends of features that extracted in the feature extracting state. in addition, proposed method should keep threshold between speed and accuracy. in this paper we proposed a two step detection method, to protect android application against repackaging attack. proposed method reduce pairwise comparison space by developing an approach based on k-nearest neighbor classifier. in addition, pairwise comparison done by set of code based features extracted from each application like API calls, method calls and etc. to detect the repackaged application of on app, k-nearest neighbor extracted and similarity between them computed by fuzzy hashing techniques. By evaluating the proposed method on 1181 application and 1196 pairs, we enhanced the state of art method, regarding speed and accuracy. for more detail, speed in repackaged android application increased 6 time against previous work with same recall but 1 percent less precision.

Keywords: Android,Repackaged,Detection,Method

Sharif University of Technology

Department of Computer Engineering

M.Sc. Thesis

Performance Improvement of Android Repackaged Applications

By:

Mojtaba Moazen

Supervisor:

Dr. Amini

february 2023