



دانشگاه صنعتی شریف
دانشکده مهندسی کامپیوتر

پایان نامه کارشناسی ارشد
مهندسی رایانش امن

بهبود کارایی روش های تشخیص برنامه های اندرویدی باز بسته بندی شده

نگارش

مجتبی موذن

استاد راهنما

دکتر مرتضی امینی

بهمن ۱۴۰۱

به نام خدا
دانشگاه صنعتی شریف
دانشکده مهندسی کامپیوتر

پایان نامه کارشناسی ارشد

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

عنوان: بهبود کارایی روش های تشخیص برنامه های اندرویدی باز بسته بندی شده

نگارش: مجتبی مودن

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 همکاران عزیزی که با راهنمایی‌های خود در بهبود نگارش این نوشتار سهیم بوده‌اند، صمیمانه سپاسگزارم.

چکیده

با گسترش روزافزون استفاده از برنامه‌های اندرویدی در سالیان اخیر حملات موجود بر روی این سیستم عامل با افزایش قابل توجهی همراه بوده است. متن باز بودن برنامه‌های اندرویدی و در نتیجه، دسترسی به کد منبع این دسته از برنامه‌ها، در کنار افزایش حملات بر روی آن‌ها، لزوم توجه به مقابله با حملات مطروحه در این زمینه را افزایش داده است. حملات بازبسته‌بندی روی برنامه‌های اندرویدی، نوعی از حملات هستند که در آن مهاجم، پس از دسترسی به کد منبع برنامه و کپی کردن آن و یا ایجاد تغییراتی که مدنظر مهاجم است، مجدداً آن را بازبسته‌بندی می‌کند. تغییر کدهای برنامه، اهداف متفاوتی نظیر تغییر کتابخانه‌های تبلیغاتی، نقض امنیت کاربر و یا ضربه به شرکت‌های تولید برنامه از تغییر گسترش برنامه‌های جعلی را دنبال می‌کند. بازبسته‌بندی برنامه‌های اندرویدی علاوه بر ماهیت تهدید کاربران و شرکت‌ها، ماهیتی پیشگیرانه نیز دارد. در این حالت توسعه‌دهندگان نرم‌افزار از طریق ایجاد مبهم‌نگاری در برنامه‌های اندرویدی، سعی در پیشگیری از بازبسته‌بندی به وسیله‌ی مهاجمان دارند. تشخیص بازبسته‌بندی در برنامه‌های اندرویدی از آن جهت دارای اهمیت است که هم کاربران و هم شرکت‌های توسعه‌دهنده، می‌توانند از این موضوع ذی‌نفع باشند. تشخیص برنامه‌های بازبسته‌بندی شده، به جهت چالش‌های پیش‌رو، نظیر مبهم‌نگاری کدهای برنامه 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 برنامه و یا طبقه‌بندی برنامه‌های موجود استفاده کرده‌اند. از طرفی برقراری حد واسطی میان سرعت و دقت در تشخیص برنامه‌های جعلی، چالشی است که استفاده از این دست پژوهش‌ها را در یک محیط صنعتی ناممکن ساخته است. در این پژوهش پس از استخراج کدهای برنامه به وسیله‌ی چارچوب سوت و ابزارهای دیس‌اسمبل، در یک روش دو مرحله‌ای کدهای برنامه‌های موجود با یکدیگر مقایسه می‌شود. پس از دیس‌اسمبل کدهای هر برنامه، در طی یک فرایند طبقه‌بندی مبتنی بر ویژگی‌های انتزاعی و دیداری، برنامه‌های کاندید برای هر برنامه مبدا استخراج می‌شود. سپس برای هر کلاس برنامه اندرویدی، امضایی متشکل از مهم‌تری 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، برنامه بازبسته‌بندی شده مشخص می‌شود. در قسمت آزمون روش پیشنهادی در این پژوهش، توانستیم روش موجود در این زمینه را با بهبود امضای تولیدشده از هر برنامه و اضافه‌شدن مرحله‌ی پیش‌پردازش، سرعت تشخیص را ۴ برابر افزایش داده و در عین حال دقت روش موجود را نیز حفظ کنیم.

کلیدواژه‌ها: پایان‌نامه، حروف چینی، قالب، زی‌پرشین

فهرست مطالب

| | | |
|----|--|---|
| ۱ | مقدمه | ۱ |
| ۷ | مفاهیم اولیه | ۲ |
| ۷ | ۱-۲ مبهم سازی | ۲ |
| ۷ | ۲-۱-۱ روش های ساده | ۲ |
| ۸ | ۲-۱-۲ روش های میانی | ۲ |
| ۱۰ | ۲-۱-۳ روش های خاکستری | ۲ |
| ۱۱ | ۲-۱-۴ روش های ترکیبی | ۲ |
| ۱۱ | ۲-۱-۵ انواع مبهم نگارها | ۲ |
| ۱۲ | ۲-۲ ساختار فایل های برنامه های اندرویدی | ۲ |
| ۱۴ | ۲-۳ کتابخانه های اندرویدی | ۲ |
| ۱۴ | ۲-۴ طبقه بندی | ۲ |
| ۱۵ | ۲-۵ بازبسته بندی برنامه های اندرویدی | ۲ |
| ۱۶ | ۳ تعریف مسئله و مرور کارهای پیشین | ۳ |
| ۱۷ | ۳-۱ تعریف مسئله | ۳ |
| ۱۸ | ۳-۲ روند کلی تشخیص برنامه های بازبسته بندی شده | ۳ |
| ۱۸ | ۳-۲-۱ پیش پردازش برنامه های اندرویدی | ۳ |
| ۱۹ | ۳-۲-۲ استخراج ویژگی | ۳ |

| | |
|----|---|
| ۲۰ | تشخیص بازبسته‌بندی ۳-۲-۳ |
| ۲۱ | روش‌های تشخیص بازبسته‌بندی ۳-۳ |
| ۲۱ | مبتنی بر تحلیل ایستا ۳-۳-۱ |
| ۳۳ | مبتنی بر تحلیل پویا ۳-۳-۲ |
| ۳۶ | سایر روش‌ها ۳-۳-۳ |
| ۳۶ | پیش‌گیری از بازبسته‌بندی ۳-۴ |
| ۳۷ | روش‌های مبتنی بر نشان‌گذاری ۳-۴-۱ |
| ۳۹ | روش‌های مبتنی بر مبهم‌نگاری ۳-۴-۲ |
| ۴۲ | مقایسه‌ی روش‌ها ۳-۵ |
| ۴۴ | نتایج جدید ۴ |
| ۴۵ | نتیجه‌گیری ۵ |
| ۴۶ | نتیجه‌گیری ۶ |
| ۴۷ | مراجع |
| ۵۷ | واژه‌نامه |
| ۵۹ | مطالب تکمیلی آ |

فهرست جدول‌ها

۳-۱ ۱۰ منبع پربازدید در پژوهش [۱] ۳۱

فهرست شکل‌ها

- ۸ ۲-۱ نمونه‌ای از مبهم‌نگاری با استفاده از تغییر نام شناسه‌ها
- ۱۰ ۲-۲ نمونه‌ای از مبهم‌نگاری با استفاده از قابلیت بازتاب به منظور پنهان‌سازی واسطه فراخوانی شده به نام batteryinfo
- ۱۳ ۲-۳ ساختار پوشه‌ها و فایل‌های بسته‌های Apk [۲]
- ۲۶ ۳-۱ مراحل تشخیص برنامه‌های بازبسته‌بندی شده در پژوهش آقای آلدینی
- ۲۷ ۳-۲ هر گره از گراف در پژوهش نگویان شامل لیستی از واسطه‌های فراخوانی شده در آن فعالیت است. [۳]

فصل ۱

مقدمه

سیستم عامل^۱ اندروید^۲ به دلیل سهولت در توسعه^۳ توسط توسعه دهندگان^۴ موبایلی و در نتیجه فراوانی استفاده از آن در تلفن های همراه، تلوزیون های هوشمند و دیگر دستگاه های موجود، حجم بالایی از بازار مصرفی سیستم عامل های موبایلی را به خود اختصاص داده است. بر طبق گزارش پایگاه استاتیسیتا^۵ [۴] سیستم عامل اندروید سهمی معادل ۷۱ درصدی از سیستم عامل های موبایلی را در سه ماهه ی پایانی سال ۲۰۲۲ به خود اختصاص داده است. در سال های اخیر به دلیل گسترش استفاده از این بستر^۶، فروشگاه های اندرویدی زیادی به جهت ارائه ی خدمات به کاربران به وجود آمده است. برخی از فروشگاه های رسمی مانند فروشگاه اندرویدی گوگل^۷، از ابزارهایی نظیر پلی پروتکت^۸ [۵] برای بررسی برنامه های اندرویدی موجود در فروشگاه استفاده می کنند. علاوه بر این، در سال های اخیر فروشگاه های متعدد رایگانی به وجود آمده اند که صرفاً برنامه های اندرویدی موجود در سطح وب را غربان^۹ و آن را به کاربران ارائه می دهند. فروشگاه های رایگان غالباً ابزارهای مشخصی را برای حفظ امنیت کاربران استفاده نمی کنند و امنیت کاربران این دسته از فروشگاه های اندرویدی، همواره تهدید می شود. یکی از راه های مورد استفاده توسط مهاجمان برای وارد ساختن بدافزار^{۱۰} به تلفن های همراه، بازبسته بندی نرم افزار^{۱۱} است. مطابق تعریف، بازبسته بندی شامل

Operation System^۱

Android^۲

Development^۳

Developers^۴

Statista^۵

Platform^۶

Google^۷

Play Protect^۸

Crawl^۹

Malware^{۱۰}

Software Repackaging^{۱۱}

دانلود^{۱۲} یک برنامه، دسترسی به محتوای کدهای برنامه اصلی از طریق روش‌های مهندسی معکوس^{۱۳} و در نهایت بازبسته‌بندی به همراه تغییر و یا بدون تغییر دادن کدهای برنامه اصلی^{۱۴} است. زبان اصلی توسعه در برنامه‌های اندرویدی، زبان جاوا^{۱۵} می‌باشد که یک زبان سطح بالا^{۱۶} محسوب می‌شود. در طی فرآیند کامپایل^{۱۷} برنامه‌های اندرویدی، مجموعه‌ی کدهای منبع در طی فرایندی به بایت‌کدهای دالویک^{۱۸} تبدیل می‌شوند و در ادامه ماشین مجازی جاوا^{۱۹}، بایت‌کدها را بر روی ماشین مقصد اجرا می‌کند [۶]. فهم و در نتیجه مهندسی معکوس زبان میانی دالویک بایت‌کدها آسان است و به همین علت موجب سهولت در بازبسته‌بندی برنامه‌های اندرویدی می‌شود.

به طور کلی بازبسته‌بندی را می‌توان از دو جهت مورد بررسی قرار داد، از دید توسعه‌دهندگان، بازبسته‌بندی شامل فرایندی است که توسعه‌دهنده با انجام مبهم‌نگاری^{۲۰} در برنامه مورد توسعه، فهم بدنه‌ی اصلی برنامه را برای مهاجم^{۲۱} سخت می‌کند. از این دید، بازبسته‌بندی یک روش تدافعی تلقی می‌شود تا مهاجم پس از دسترسی به کد برنامه اصلی، نتواند بدنه‌ی برنامه اصلی برنامه را شناسایی و در نتیجه آن را تغییر دهد. از جهت دیگر، بازبسته‌بندی توسط فردی که برنامه متعلق به او نیست یک عمل تهاجمی محسوب می‌شود. در این حالت، مهاجم پس از دسترسی به کد برنامه اصلی، بسته به هدف او، برنامه را مجدداً بازبسته‌بندی می‌کند و آن را در فروشگاه‌های اندرویدی خصوصاً فروشگاه‌هایی که نظارت کمتری بر روی آن‌ها وجود دارد منتشر می‌کند. در دیدگاه تهاجمی، مهاجم به جهت اهدافی متفاوتی نظیر تغییر کدهای تبلیغاتی^{۲۲} در برنامه 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 [۷]. مطابق پژوهش آقای ژو و همکاران [۸] حدود ۸۵ درصد بدافزارهای موجود، از طریق بازبسته‌بندی منتشر می‌شوند. همانطور که گفته شد، برخی فروشگاه‌های اندرویدی نظیر گوگل، سازوکار مشخصی را برای تشخیص^{۲۳} بازبسته‌بندی ارائه‌داده‌اند اما بسیاری از فروشگاه‌های اندرویدی فعال و پربازدید، خصوصاً فروشگاه‌های رایگان، یا از هیچ ابزاری استفاده نمی‌کنند و یا در صورت توسعه‌ی نرم‌افزار بومی^{۲۴} خود برای شناسایی برنامه‌های بازبسته‌بندی شده، مشخصات و یا دقت آن را گزارش نکرده‌اند.

همانطور که اشاره شد، به دلیل محبوبیت و در نهایت استفاده‌ی زیاد برنامه‌های اندرویدی و همچنین

- Download^{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^{۲۴}

نظارت کم در فروشگاه‌های مرتبط، بازبسته‌بندی یک روش پر استفاده به جهت انتقال بدافزار به تلفن همراه کاربران است. آقای خانمحمدی و همکاران [۹]، پس از بررسی برنامه‌های اندرویدی مجموعه داده‌ی^{۲۵} اندروزو^{۲۶}، دریافتند که ۵۲/۲۲٪ از برنامه‌های موجود در این مخزن توسط ویروس‌توتال^{۲۷}، بدافزار شناسایی شده‌اند. ویروس‌توتال، ابزاری متشکل از ۳۰ ضدبدافزار برای بررسی یک برنامه اندرویدی است. مطابق این پژوهش، ۷۷/۸۴٪ از برنامه‌های این مجموعه داده که بازبسته‌بندی شده‌اند، دارای نوعی از بدافزار ضدتبلیغاتی^{۲۸} بوده‌اند که موجب می‌شود تبلیغات موجود در برنامه تغییر کرده و اهداف مالی و امنیتی کاربران و توسعه‌دهندگان مخدوش شود. علاوه بر این، مطابق پژوهشی که توسط ویداس و همکاران [۱۰] انجام شده‌است، پس از پیاده‌سازی ۷ روش پربازدید به جهت تشخیص بازبسته‌بندی، در بهترین حالت، روش‌های موجود قادر به تشخیص ۷۲/۲۲٪ از برنامه‌های بازبسته‌بندی شده‌ی سه فروشگاه مطرح اندرویدی بوده‌اند. بنابراین مشخص است که تشخیص برنامه‌های بازبسته‌بندی شده، به چه میزان می‌تواند اهداف مالی و امنیتی توسعه‌دهندگان و کاربران برنامه‌ها را ارضا کند. در سال‌های اخیر ارائه‌ی یک راهکار پرسرعت به همراه دقت مناسب، همواره یکی از دغدغه‌های مهم پژوهش‌کنندگان در این زمینه بوده‌است.

همانطور که گفته‌شد، بازبسته‌بندی برنامه‌های اندرویدی از دو دیدگاه تهاجمی و تدافعی قابل بررسی است. در حالتی که کاربر متقلب، برنامه اندرویدی اصلی را دچار تغییراتی می‌کند و آن را در اختیار عموم قرار می‌دهد، تشخیص بازبسته‌بندی، با استفاده از مقایسه‌ی برنامه اصلی و برنامه 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 برنامه مبدا و مقصد پیاده‌سازی می‌کند. عمده‌ی روش‌های موجود در این طبقه مبتنی بر تحلیل ایستا^{۲۹}ی جفت برنامه‌ها است و استفاده از تحلیل پویا^{۳۰} به جهت سرعت پایین آن، محبوبیت فراوانی ندارد [۱۱]. در سمت دیگر طبقه‌بندی^{۳۱} برنامه‌های اندرویدی وجود دارد. روش‌های موجود در این دسته، عمدتاً سرعت بالایی دارند اما در تشخیص جفت بازبسته‌بندی شده دقت پایینی را ارائه می‌دهند.

برنامه‌های اندرویدی متشکل از دو قسمت اصلی کدهای برنامه و منابع^{۳۲} هستند. کدهای برنامه، منطق^{۳۳} برنامه را تشکیل می‌دهند و رفتار برنامه با توجه به این قسمت مشخص می‌شود. از طرفی منابع

- Data Set^{۲۵}
- Androzoo^{۲۶}
- Virus total^{۲۷}
- AdWare^{۲۸}
- Static^{۲۹}
- Dynamic^{۳۰}
- Classification^{۳۱}
- Resources^{۳۲}
- Logic^{۳۳}

برنامک، رابط کاربری^{۳۴} 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 حالتی 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 گاهی بدون هیچ تغییری در کد برنامک، ادعای توسعه‌ی یک برنامک جدید را اثبات کند. لازم به ذکر است استفاده از ویژگی‌های کدپایه^{۳۵} و منبع‌پایه^{۳۶}، به وفور در پژوهش‌های سال‌های اخیر یافت می‌شود که هر کدام معایب و مزایای خود را دارد.

در روش‌های مبتنی بر طبقه‌بندی عمدتاً تعریف تشخیص بازبسته‌بندی محدود به تشخیص دسته‌ی مشکوک که احتمال بازبسته‌بندی بودن جفت‌های داخل این دسته، بیش از سایر دسته‌ها است. بنابراین 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، دوجالش مهم در راستای تشخیص بازبسته‌بندی است. متقلب پس از بازبسته‌بندی برنامک، با استفاده از مبهم‌نگاری 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^{۳۷}یی که در برنامک مورد استفاده قرار گرفته‌اند به همراه کد مورد توسعه، کامپایل شده و دالویک بایت‌کدهای آن در کنار برنامک قرار می‌گیرد. بر اساس پژوهش آقای زیانگ و همکاران [۱۲] 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^{۳۸} و یا روش تشخیص

^{۳۴}User Interface

^{۳۵}Code Base

^{۳۶}Resource Base

^{۳۷}Library

^{۳۸}White List

مبتنی بر شباهت سنجی^{۳۹}. در روش لیست سفید، لیستی از مشهورترین کتابخانه‌های موجود در مخازن کتابخانه‌ای اندروید نظیر ماون^{۴۰} جمع آوری می‌شود و با استفاده از نام کلاس‌ها و بسته‌های موجود، کلاس‌های کتابخانه‌ای تشخیص داده 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 برنامه‌ی مبدا و مخزن کتابخانه‌های اندروید صورت می‌گیرد و در نهایت از طریق شباهت سنجی، کدهای کتابخانه‌ای تشخیص داده می‌شوند. مشخص است که روش‌های مبتنی بر شباهت سنجی از دقت بیشتری، خصوصاً 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 برنامه‌های 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 استفاده از ویژگی‌هایی را که مقاومت بالایی مقابل مبهم‌نگاری داشته باشند واجب کند. از طرفی استفاده از ویژگی‌های مقاوم به مبهم‌نگاری، می‌تواند سرعت تشخیص را بسیار پایین آورده تا حدی که عملاً 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 شباهت سنجی، روشی را ارائه کرده‌ایم که در حالی که مقاومت بالایی نسبت به مبهم‌نگاری داشته باشد، سرعت روش‌های پیشین را نیز افزایش دهد. در این پژوهش به عنوان پیش‌پردازش، از یک طبقه‌بند^{۴۱} نزدیک‌ترین همسایه^{۴۲} برای کاهش فضای مقایسه‌ی دودویی^{۴۳} و با استفاده از ویژگی‌های مبتنی بر منبع، سرعت تشخیص بهبود داده شده است. با کاهش فضای مقایسه‌ی دودویی و طبقه‌بندی برنامه‌های مشکوک در یک دسته، مقایسه‌ی برنامه‌های موجود در آن دسته آغاز می‌شود. مقایسه‌ی دودویی در هر دسته مبتنی بر تحلیل ایستا و شباهت سنجی کدهای برنامه‌ی انجام می‌شود. ابتدا ویژگی‌هایی از هر کلاس و متد^{۴۴} در بسته‌های برنامه‌ی استخراج شده و امضا^{۴۵}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 برنامه‌های بازبسته‌بندی شده است. حذف کدهای کتابخانه‌ای با استفاده از روشی مبتنی بر مقایسه‌ی کدهای موجود در مخزن کتابخانه‌ها و کلاس‌های برنامه‌ی انجام می‌شود. مخزن

Similarity^{۳۹}

Maven Repository^{۴۰}

Classifier^{۴۱}

Nearest Neighbor^{۴۲}

Pairwise Comparison^{۴۳}

Method^{۴۴}

Signature^{۴۵}

کتابخانه‌ها متشکل از ۴۵۳ کتابخانه‌ی اندرویدی جمع‌آوری شده از مخزن ماون می‌باشد. در نهایت پس از تشخیص کلاس‌های کتابخانه‌های اندرویدی و حذف آن‌ها از کد برنامه، کدهای مورد توسعه به عنوان ورودی برای مقایسه‌ی دودویی و طبقه‌بندی مورد تحلیل قرار می‌گیرند.

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 خواهد شد 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 خواهیم پرداخت.

فصل ۲

مفاهیم اولیه

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

۱-۲ مبهم‌سازی

آن‌چنان که در فصل پیشین گفته شد، 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، برنامه‌ک اصلی را دانلود کرده و پس از دیکامپایل^۱ کردن، به نوعی تغییر می‌دهد که منطق کلی برنامه، تغییری نمی‌کند. مبهم‌سازی یکی از ارکان اصلی در فرایند بازبسته‌بندی است و هدف اصلی آن این است که ابزارهای تشخیص بازبسته‌بندی، خصوصاً در مواردی که از تحلیل ایستا استفاده می‌کنند را به اشتباه بیاندازد. روش‌های مبهم‌سازی را از نظر میزان سختی در تشخیص به ۳ دسته‌ی کلی می‌توان تقسیم کرد[۱۳]:

۱-۱-۲ روش‌های ساده

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

^۱Decompile

باز بسته‌بندی تنها موجب تغییر در امضاء توسعه‌دهنده‌ی برنامه‌ی و جمع‌آزمای^۲ می‌شود. بنابراین روش‌هایی که مبتنی بر این دو خصوصیت هستند در این سطح دچار مشکل می‌شوند.

۲-۱-۲ روش‌های میانی

این دسته از روش‌های مبهم‌سازی، شامل روش‌هایی است که در آن بیشتر ویژگی‌های مبتنی بر معناسازی^۳ تغییر می‌کند و ویژگی‌های مبتنی بر نحو^۴ ثابت باقی می‌مانند. بنابراین، روش‌هایی که بیشتر مبتنی بر معناسازی برنامه‌های اندرویدی هستند، دچار خطای بیشتری در این سطح از مبهم‌نگاری می‌شوند. در ادامه به معرفی مختصری از انواع روش‌های مبهم‌نگاری مطابق با پژوهش [۱۴] در این دسته می‌پردازیم:

- **تغییر نام شناسه‌ها^۵:** تغییر نام شناسه‌های موجود در برنامه شامل نام کلاس‌ها، متدها و یا متغیرها^۶ی موجود [۱۳]

```
1 public class a{
2     private Integer a;
3     private Float b;
4     public void a(Integer a, Float b){
5         this.a = a + Integer.valueOf(b)
6     }
7 }
```

شکل ۲-۱: نمونه‌ای از مبهم‌نگاری با استفاده از تغییر نام شناسه‌ها

- **تغییر نام بسته:** در این روش مبهم‌نگاری با استفاده از تغییر نام بسته‌های برنامه صورت می‌گیرد.
- **رمزنگاری رشته‌ها:** استفاده از رمزنگاری در رشته‌های^۷ مورد استفاده از فایل‌های دکس^۸، موجب کاهش سطح معناسازی می‌شود.
- **فراخوانی غیرمستقیم:** یکی از روش‌های ساده‌ی تغییر گراف فراخوانی^۹، استفاده از یک تابع واسط به عنوان تابع فراخوانده‌ی^{۱۰} تابع اصلی است. در این حالت تابع اولیه یک تابع واسط و تابع واسط به صورت زنجیره‌ای تابع اصلی را فراخوانی می‌کند. بدنه‌ی تابع واسط در این حالت، بسیار ساده و شامل یک فراخوانی تابع اصلی است.

Checksum^۲
Semantic^۳
Syntax^۴
Identifier^۵
Variable^۶
String^۷
Dex Files^۸
Call Graph^۹
Caller^{۱۰}

فقره جابه‌جایی دستورات: جابه‌جایی دستورات موجود در برنامه اصلی، یکی از روش‌های پرکاربرد توسط ابزارهای مبهم‌نگاری است. جابه‌جایی دستورات به شکلی انجام می‌شود که استقلال هر قسمت حفظ گردد.

- **جابه‌جایی ساختار سلسله‌مراتبی:** در این روش، ساختار سلسله‌مراتبی کلاس‌های برنامه به نوعی تغییر می‌کند که منطق کلاس‌ها دچار تغییر نشود.

- **ادغام و شکستن:** می‌توان توابع و یا کلاس‌های موجود در برنامه‌های اندرویدی را ادغام کرد. برای مثال می‌توان هر جایی که یک تابع صدا زده شده 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 برنامه را تغییر داد.

- **وارد ساختن کدهای بیهوده:** کدهای بیهوده، کدهایی هستند که اجرا می‌شوند ولی تاثیری در ادامه‌ی روند اجرایی برنامه، ندارند. کدهای بیهوده عموماً دارای ساختارهای کنترلی^{۱۱} و حلقه‌های نپ^{۱۲} هستند که تاثیری در روند اجرای برنامه ندارند. ذکر این نکته حائز اهمیت است که در صورتی که در ساختار کدهای بیهوده از شروط کنترلی مبتنی بر متغیرهای پویا^{۱۳} استفاده شود آنگاه دیگر تحلیل ایستای برنامه‌های اندرویدی قادر به تشخیص این نوع از مبهم‌نگاری‌ها نیست.

- **وارد ساختن کدهای مرده:** یکی دیگر از روش‌های تغییر گراف‌های برنامه از جمله گراف جریان^{۱۴}، اضافه‌کردن کدهای مرده‌ای است که در ساختار گراف جریان برنامه‌های اندرویدی هیچ‌گاه اجرا نمی‌شوند اما به عنوان یک گره در گراف حضور دارند.

- **روش‌های دیگر:** روش‌های دیگری نظیر تغییر نام منابع مورد استفاده در برنامه‌های اندرویدی و حذف فایل اشکال‌زدایی^{۱۵} از روش‌های دیگری است که در این سطح به وفور مورد استفاده قرار می‌گیرد.

^{۱۱} Control's Statement

^{۱۲} Nop

^{۱۳} Dynamic Variables

^{۱۴} Flow Graph

^{۱۵} Debug File

۳-۱-۲ روش‌های خاکستری

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

- **بازتاب^{۱۶}**: بازتاب یکی از ویژگی‌های مهم و پیچیده‌ی زبان جاوا می‌باشد [۱۵] که امکان فراخوانی متدها و ارتباط با کلاس‌های برنامه 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 برنامه‌های اندرویدی ایجاد سازند. استفاده از قابلیت بازتاب و رمزنگاری^{۱۷} 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 ۲-۲: 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 برنامه بازبسته‌بندی شده، قسمتی مهمی از کدهای برنامه را رمزنگاری کرده و در هنگام اجرا با استفاده از یک رویه‌ی رمزگشایی^{۱۸}، کدهای اصلی را بارگیری^{۱۹} 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^{۲۰}**: زبان جاوا از قابلیت مهمی به نام بارگیری پویای کد پشتیبانی می‌کند که اجازه می‌دهد تکه کدی را که پیش از این در کد مورد توسعه‌ی یک برنامه موجود نبوده را در حین اجرا به برنامه اضافه کنیم. مهاجم با استفاده از این قابلیت زبان جاوا می‌تواند قسمت‌هایی

^{۱۶} Reflect
^{۱۷} Encryption
^{۱۸} Decryption
^{۱۹} Load
^{۲۰} Dynamic Class Loading

از برنامه‌ک را در حین اجرای آن تغییر دهد که عملاً تشخیص آن‌ها با استفاده از تحلیل‌های ایستا امکان‌پذیر نیست.

۴-۱-۲ روش‌های ترکیبی

هر ترکیبی از روش‌های گفته‌شده در سطوح مختلف را می‌توان برای مبهم‌نگاری استفاده کرد. به صورت کلی روش‌های میانی ۲-۱-۲ و روش‌های خاکستری ۳-۱-۲ 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.

۵-۱-۲ انواع مبهم‌نگارها

در قسمت پیشین، دریافتیم که مبهم‌نگاری، سطوح متفاوتی دارد که متقلبان برای تولید برنامه‌های بازبسته‌بندی شده از آن‌ها استفاده می‌کنند. از آنجایی که بسیاری از برنامه‌های تقلبی با استفاده از مبهم‌نگار^{۲۱}ها توسعه یافته‌اند و علاوه بر این برای ابداع یک روش مفید جهت تشخیص برنامه‌های بازبسته‌بندی شده ابتدا باید انواع مبهم‌نگارهای موجود را بررسی کرد. در پژوهشی که توسط ژانگ و همکاران [۱۶] انجام شده، ۴۳٪ از برنامه‌های بازبسته‌بندی شده‌ی مورد بررسی در این پژوهش، از مبهم‌نگاری‌های بسیار ساده‌ای نظیر تغییر نام و با استفاده از مبهم‌نگارهای رایگان، انجام شده‌است. در ادامه به بررسی چند مبهم‌نگار رایگان و تجاری^{۲۲} می‌پردازیم.

• پروگارد

پروگارد^{۲۳} 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 از روش‌های مشروحه در بخش ۳-۱-۲ انجام می‌شود. [۱۷]

• آلاتوری

آلاتوری^{۲۴} یک مبهم‌نگار رایگان تولیدشده توسط شرکت روسی اسماردک^{۲۵} می‌باشد که سطوح مختلفی از مبهم‌نگاری را با توجه به فایل‌های پیکربندی^{۲۶} پوشش می‌دهد. این مبهم‌نگار از تغییرنام،

^{۲۱} Obfuscator

^{۲۲} Commercial

^{۲۳} Proguard

^{۲۴} Allatori

^{۲۵} Smardec

^{۲۶} Configuration

مبهم‌نگاری مبتنی بر تغییر گراف‌های جریان، مبهم‌نگاری فایل‌های اشکال‌زدایی و رمزنگاری داده‌های رشته‌ای پشتیبانی می‌کند. [۱۸، ۱۹]

• دکس‌گارد

این مبهم‌نگار نسخه‌ی تجاری نرم‌افزار پروگارد است که توسط شرکت گارداسکووار^{۲۷} تولید شده‌است. دکس‌گارد^{۲۸} 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.

۲-۲ ساختار فایل‌های برنامه‌های اندرویدی

هر برنامه‌ی اندرویدی یک فایل فشرده‌شده با پسوند APK^{۲۹} است که به اختصار شامل ۴ پوشه‌ی مهم و ۳ فایل می‌باشد. برای درک بهتر از روش پیشنهادی این پژوهش، در ادامه هر کدام از این قسمت‌ها را معرفی و کارکرد آن را بررسی خواهیم کرد. [۲۰]

• **پوشه‌ی res:** این پوشه شامل منابع برنامه‌های اندرویدی است که مربوط به رابط کاربری برنامه می‌شود. این پوشه در نهایت به فایل‌های R. نگاشت شده و هر کدام از منابع با یک شناسه^{۳۰} مشخص می‌گردد.

• **پوشه‌ی lib:** فایل‌های کامپایل‌شده‌ی بومی در این پوشه قرار می‌گیرند که شامل کتابخانه‌های اندرویدی و جاوایینیز می‌گردد. استفاده از فایل‌هایی که کامپایل شده‌اند سرعت اجرای برنامه‌های اندرویدی را بالا می‌برد لذا استفاده از آن‌ها به عنوان بسته‌های^{۳۱} از پیش آماده محبوبیت دارد.

• **فایل Classes.dex:** فایل‌های با پسوند dex فایل‌های دودویی^{۳۲} هستند که اطلاعات را در سطر و ستون‌های خود ذخیره می‌کنند. این فایل در برنامه‌های اندرویدی حاوی بایت‌کدهای دالویک است که توسط ماشین مجازی دالویک^{۳۳} اجرا می‌شود.

^{۲۷} Guardsquare

^{۲۸} DexGuard

^{۲۹} Android Package

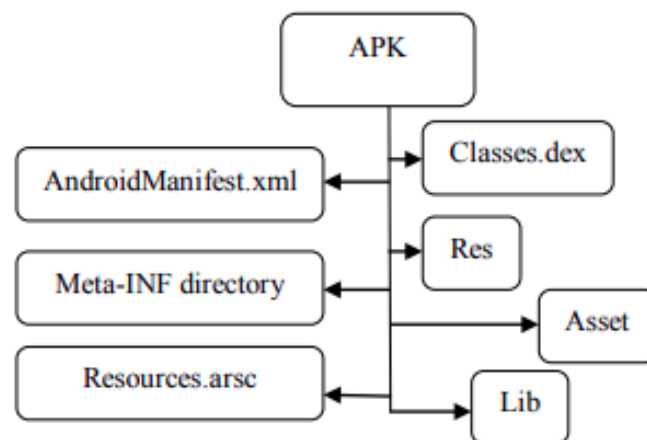
^{۳۰} Resource Id

^{۳۱} Module

^{۳۲} Binary

^{۳۳} Dalvik Virtual Machine

- فایل **AndroidManifest.xml**: پیکربندی‌های مهم فایل‌های APK از جمله لیست مجوزهای مورد نیاز، لیست مولفه‌ها^{۳۴} و نام بسته‌ی برنامه 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^{۳۵} کاربر توسعه‌دهنده‌ی برنامه است که برنامه با کلید خصوصی متناظر آن امضا شده‌است. امضای موجود در این پوشه، خاصیت صحت‌سنجی^{۳۶} دارد اما اطلاعاتی را از توسعه‌دهنده نشر نمی‌دهد و به صورت خودامضا ساخته می‌شود.
- فایل **resources.arsc**: این فایل برای انجام نگاشت^{۳۷} میان منابع موجود در پوشه‌ی resources و شناسه‌ی هر منبع استفاده می‌شود تا بتوان در حین اجرای برنامه‌ها، هر شناسه را به منبع آن ترجمه کرد.



شکل ۲-۳: ساختار پوشه‌ها و فایل‌های بسته‌های Apk [۲]

^{۳۴}Component
^{۳۵}Public Key
^{۳۶}Integrity
^{۳۷}Mapping

۳-۲ 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 دو بخش کتابخانه‌های مختص برنامه‌نویسی اندرویدی و کتابخانه‌های زبان جاوا تقسیم می‌شوند. در هنگام کامپایل، تمامی کتابخانه‌هایی که توسعه‌دهنده هنگام توسعه‌ی برنامه، آن‌ها را استفاده کرده‌است به همراه کدهای مورد توسعه، کامپایل شده و در ساختار سلسله‌مراتبی تحت فایل‌های `classes.dex` قرار می‌گیرد [۲۱]. ذکر این نکته قابل توجه است که تشخیص برنامه‌های بازبسته‌بندی شده بدون شناسایی کتابخانه‌های برنامه اندرویدی مورد نظر امکان‌پذیر نیست. واضح است که در صورتی که نتوانیم کتابخانه‌های اندرویدی شاخص را از جفت برنامه‌های مورد بررسی جدا کنیم، آنگاه بخش زیادی از شباهت دو برنامه ناشی از کتابخانه‌های اندرویدی و اشتراکات موجود در آن‌ها است چرا که بسیاری از کتابخانه‌ها خصوصاً کتابخانه‌های زبان جاوا، در هر برنامه 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 برنامه‌های بازبسته‌بندی در این حوزه شده‌است.

۴-۲ طبقه‌بندی

طبقه‌بندی اطلاعات ورودی یکی از روش‌های مرسوم در هوش مصنوعی و یادگیری ماشین است که توسط الگوریتم‌های طبقه‌بند انجام می‌شود. یک طبقه‌بند شامل مجموعه‌ای از الگوریتم^{۳۸}‌ها است که برای طبقه‌بندی و یا مرتب‌سازی^{۳۹} داده‌های ورودی مورد استفاده قرار می‌گیرد [۲۲]. یکی از ساده‌ترین مثال‌های موجود برای طبقه‌بندی، جداسازی هرزنامه^{۴۰}‌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.

Algorithm^{۳۸}
Sorting^{۳۹}
Spam^{۴۰}

۵-۲ بازبسته‌بندی برنامه‌های اندرویدی

با پوشش عمیق در پژوهش‌های مرتبط با این حوزه در سالیان اخیر متوجه می‌شویم که تعاریف متنوعی برای بازبسته‌بندی در نظر گرفته شده است. برخی از پژوهش‌ها نظیر [۲۳، ۲۴] بازبسته‌بندی را در تغییر منابع و ظاهر برنامه‌ها در نظر می‌گیرند و در نهایت ویژگی‌های مبتنی بر ظاهر آن‌ها را با یکدیگر مقایسه می‌کنند در حالی که برخی از پژوهش‌های اخیر دیگر نظیر [۲۵، ۲۶] بازبسته‌بندی مبتنی بر تغییر ویژگی‌های کدپایه تعریف شده است. البته که نمی‌توان به هیچ کدام از تعاریف بالا خرده گرفت چرا که هر دو تعریف از نظر مهاجم و اهداف تعریف شده توسط اون قابل استناد است. علاوه بر این یکی دیگر از اختلافات موجود در تعریف بازبسته‌بندی، وجود مبهم‌نگاری در برنامه‌های بازبسته‌بندی شده است. برخی از پژوهش‌ها نظیر [۲۷] بازبسته‌بندی را منوط به تغییر در امضای برنامه می‌دانند اما بسیاری از پژوهش‌های به‌روزتر، نظیر [۲۸، ۲۹] بازبسته‌بندی را تنها به تغییر منابع و یا کدهای برنامه تقلبی نسبت به برنامه اصلی می‌دانند. همانطور که مشاهده شد، هنوز تعریف مشخصی از بازبسته‌بندی در پژوهش‌ها ارائه نشده است اما به طور کلی می‌توان گفت که برنامه A بازبسته‌بندی یک برنامه دیگر است اگر تغییرات آن نسبت به برنامه مبدا محدود و با حفظ کارکرد و منابع برنامه اصلی باشد. این تعریف در این پژوهش نیز به عنوان تعریف مبنای بازبسته‌بندی در نظر گرفته شده است.

فصل ۳

تعریف مسئله و مرور کارهای پیشین

پژوهش‌های اخیر در حوزه‌ی تشخیص برنامه‌های اندرویدی بازبسته‌بندی شده نشان می‌دهد که تشخیص این دسته از برنامه‌ها 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 آن جهت اهمیت دارد که تشخیص درست آن‌ها می‌تواند مثبت غلط و منفی غلط را کاهش دهد. در بیشتر مواقع، خصوصاً 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 برنامه‌های مورد بررسی، خصوصاً 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 برنامه‌هاست که مقاومت بالایی در برابر مبهم‌نگاری داشته باشند. بدین معنا که متقلب برای تغییر این دسته از ویژگی‌ها 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 برنامه‌های بازبسته‌بندی شده مبتنی بر ویژگی‌هایی صورت گرفته است که در عین مقاومت در مقابل مبهم‌نگاری، هزینه‌ی محاسباتی تشخیص برنامه‌های بازبسته‌بندی شده را افزایش می‌دهد به طوری که استفاده از این روش‌ها را عملاً در یک محیط صنعتی غیر ممکن ساخته‌است.

با توجه به اهمیت تشخیص مبهم‌نگاری و در نهایت تشخیص برنامه‌های 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 برنامه‌های بازبسته‌بندی استفاده کرده‌اند و مزایا و معایب هر کدام را به صورت جدا بررسی خواهیم کرد. از آنجایی که هدف این پژوهش بهبود کارایی روش‌های تشخیص برنامه‌های بازبسته‌بندی شده است و تمرکز پژوهش بر روی تشخیص کدهای کتابخانه‌ای نبوده است، در ابتدا روند کلی تشخیص برنامه‌های 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 برنامه‌های بازبسته‌بندی شده می‌پردازیم و مسئله‌ی تشخیص برنامه‌های 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 خواهیم کرد.

۳-۱ تعریف مسئله

علازم 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 ۳-۱ (تشخیص بازبسته‌بندی مبتنی بر برنامه‌ی مبدا) / تشخیص بسته‌ی بازبسته‌بندی شده، یعنی تشخیص جفتی از برنامه‌های درون مخزن که دقیقاً جفت مشابه برنامه‌ی ورودی باشد. به بیان دیگر در این تعریف مشخص می‌شود که برنامه‌ی ورودی بازبسته‌بندی شده است یا خیر و در صورتی که بود، جفت برنامه‌ی آن درون مخزن نیز مشخص می‌شود.

تعریف ۳-۲ (تشخیص بازبسته‌بندی مبتنی بر تصمیم‌گیری برنامه‌ی مقصد) تشخیص بسته‌ی بازبسته‌بندی شده، یعنی مشخص کنیم برنامه‌ی ورودی بازبسته‌بندی شده است یا خیر. در این حالت تشخیص برنامه‌ی اصلی اهمیت ندارد و مسئله، تصمیم‌گیری^۱ درباره‌ی بازبسته‌بندی بودن یک برنامه‌ی ورودی است.

در سال‌های اخیر، اکثر پژوهش‌ها از یکی از تعاریف بالا برای تشخیص بازبسته‌بندی استفاده کرده‌اند. برای پاسخ به تعریف ۲، پژوهش‌هایی نظیر [۳۰، ۳۱، ۳۲] از روش‌های مبتنی بر مدل‌های یادگیری ماشین^۲

^۱ Decision
^۲ Machine Learning

برای تشخیص برنامه‌های بازسته‌بندی شده استفاده کرده‌اند. حال آن‌که پژوهش‌های مرتبط با تعریف ۱، نظیر [۳۳، ۹] بیشتر از روش‌های مقایسه‌ی دودویی و مبتنی بر شباهت‌سنجی استفاده کرده‌اند. تعریفی که در این پژوهش مورد استفاده قرار گرفته است، تعریف ۱ است. یعنی تشخیص بازسته‌بندی منوط به تشخیص جفت برنامه اصلی در مخزن برنامه‌های پژوهش می‌باشد. بنابراین در طی فرایند تشخیص به ۲ سوال اساسی پاسخ می‌دهیم:

- آیا برنامه ورودی بازسته‌بندی شده‌ی یک برنامه‌ی دیگر است؟
- در صورتی که برنامه مورد بررسی، بازسته‌بندی شده‌ی برنامه دیگری بود، آنگاه جفت بازسته‌بندی شده‌ی برنامه ورودی مشخص گردد.

۳-۲ روند کلی تشخیص برنامه‌های بازسته‌بندی شده

با بررسی پژوهش‌های صورت‌گرفته در حوزه‌ی تشخیص برنامه‌های بازسته‌بندی 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 برنامه 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:

۳-۲-۱ پیش‌پردازش برنامه‌های اندرویدی

یکی از مراحل مهم در تشخیص برنامه‌های بازسته‌بندی 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:

- **مبتنی بر لیست سفید:** در این روش، لیستی از نام بسته‌ای مشهور کتابخانه‌ای در برنامه‌های اندرویدی در دسترس است و با استفاده از نام بسته‌های موجود در برنامه، کدهای کتابخانه‌ای از

^۳ Pre process
^۴ Structural

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 آن است 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 برنامه‌ها و کدهای کتابخانه‌ای موجود در مخزن، با یکدیگر مقایسه می‌شوند و بدین طریق کتابخانه‌های اندرویدی از کدهای مورد توسعه در برنامه، 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 برنامه جدا کرد.

۳-۲-۲ استخراج ویژگی

پس از حذف کدهای کتابخانه‌ای در قسمت قبلی و انجام پیش‌پردازش‌های مورد نیاز، کدهای منبع برنامه هدف، به یک طرح کلی مدل می‌شود. به صورت کلی می‌توان روش‌های تشخیص برنامه‌های بازبسته‌بندی شده را در پژوهش‌های سالیان خیر، ناشی از تفاوت در دیدگاه در مرحله‌ی استخراج ویژگی^۵ دانست. همانطور که در شکل — مشاهده می‌شود، روش‌های تشخیص برنامه‌های 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 توضیح نمی‌دهیم. به صورت کلی، می‌توان روش‌های تشخیص برنامه‌های 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.

• **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

^۵ Feature Extracting

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 برنامه‌های 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 برنامه‌های اندرویدی صورت می‌گیرد، می‌باشد. به همین علت روش‌های موجود در این حوزه، عمدتاً برنامه‌ها را در هنگام اجرا بررسی و استخراج ویژگی عمدتاً 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 برنامه‌های اندرویدی بازبسته‌بندی شده، چگونگی پیاده‌سازی شبیه‌سازها^۸ است. بسیار از شبیه‌سازها توانایی شبیه‌سازی تمامی خدمات موجود در برنامه را ندارند و برای تحلیل دقیق‌تر نیازمند استفاده از کاربران واقعی در شبیه‌سازی و استفاده از خدمات برنامه هستند. عامل دیگری که تشخیص با استفاده از تحلیل پویا را مشکل می‌کند، این است که بسیاری از بدافزارهای توسعه‌یافته، توانایی تشخیص محیط اجرای شبیه‌سازی شده را دارند و ممکن است تمامی قابلیت‌های خود و یا بخشی از آن را به جهت دور زدن سیستم‌های تشخیص پویا، پنهان کنند.

۳-۲-۳ 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 برنامه 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 برنامه‌های موجود در مخزن مقایسه می‌کند و در نهایت برنامه بازبسته‌بندی شده را مشخص می‌کند. اکثر روش‌های مبتنی بر مقایسه‌ی دودویی، جفت برنامه اصلی را نیز مشخص می‌کنند و از تعریف ۱-۳ استفاده می‌کنند بنابراین یکی از مزیت‌های این

Execution Time^۶
Sand Box^۷
Simulator^۸

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 ۲-۳ برای تشخیص برنامه‌ک بازبسته‌بندی شده استفاده می‌کنند. بنابراین، تنها تصمیم‌گیری در مورد بازبسته‌بندی بودن یا نبودن برنامه‌ک 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.

۳-۳ روش‌های تشخیص بازبسته‌بندی

همانطور که در شکل — 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.

۱-۳-۳ مبتنی بر تحلیل ایستا

در این قسمت، روش‌های مبتنی بر تحلیل ایستا و پژوهش‌های مرتبط با آن را بررسی خواهیم کرد. 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 برنامه‌ک‌های اندرویدی بازبسته‌بندی شده پشتیبانی کرد.

روش‌های مبتنی بر آپکد و دستورات

استفاده از آپکد^۹‌های موجود در فایل‌های دالویک، یکی از روش‌های تشخیص برنامه‌ک‌های بازبسته‌بندی شده است. هدف از پژوهش آقای ژو [۳۴] و همکاران، توسعه‌ی ابزاری به نام درویدمس^{۱۰} بوده است

^۹Opcodes
^{۱۰}DroidMoss

که توسط آن مشخص شود چه تعدادی از برنامه‌های موجود در فروشگاه‌های اندرویدی غیررسمی، بازبسته‌بندی شده‌ی برنامه‌های 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 پخش کردن برنامه‌های بازبسته‌بندی شده استفاده می‌کنند. برای استخراج امضای برنامه در این پژوهش از کدهای دالویک موجود در `Classes.dex` و امضای دیجیتال برنامه‌نویس در فراداده^{۱۱} استفاده شده است. پس از جداسازی کدهای کتابخانه‌ای به وسیله‌ی لیست سفید و استخراج آپکدها از فایل‌های دالویک، از یک پنجره‌ی لغزات^{۱۲} روی آپکدها استفاده شده و در نهایت چکیده‌ی Hashی آپکدها به همراه امضای دیجیتال برنامه‌نویس، موجود در پوشه‌ی META-INF تشکیل امضای برنامه را می‌دهند. همانطور که می‌توان فهمید، فرض پژوهش این بوده است که کلید خصوصی توسعه‌دهنده لو نرفته است. در نهایت برای قسمت شباهت‌سنجی، از الگوریتم فاصله ویراشی^{۱۳} استفاده شده است. در قسمت شباهت‌سنجی از ۲۹۰۶ برنامه موجود در فروشگاه‌های رسمی استفاده شده و نتایج پژوهش نشان می‌دهد که ۵ تا ۱۳ درصد از برنامه‌های موجود در فروشگاه‌های غیر رسمی، بازبسته‌بندی شده‌ی برنامه‌های فروشگاه‌های رسمی است. در پژوهش دیگری که توسط آقای ژو [۲۶] ارائه شده است، هدف پژوهش، افزایش سرعت پژوهش قبلی با استفاده از نمونه‌های n تایی از آپکدها بوده است. در این پژوهش امضای هر برنامه متشکل از قسمتی از فراداده‌ی آن شامل فایل‌های منیفست^{۱۴} و اطلاعاتی در مورد تعداد فایل‌های برنامه، توصیفات آن و چکیده‌ی آپکدهای دستورات برنامه است. این پژوهش با استفاده از یک مرحله پیش‌پردازش شامل بررسی فایل فراداده‌ی برنامه‌های موجود، فضای جست‌وجوی دودویی برنامه‌های مورد مقایسه را کاهش می‌دهد. دنوز و همکاران [۳۵] روش دیگری را مبتنی بر شباهت‌سنجی روی آپکدها با استفاده از فاصله‌ی فشرده‌سازی نرمال شده 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 بکتاب هستند از هر دو برنامه، بر اساس معیار فاصله‌ی فشرده‌سازی نرمال شده با یکدیگر مقایسه و بدین ترتیب متدهای مشابه استخراج می‌شود. در پژوهش [۳۶، ۳۷] پس از استخراج هیستوگرام‌های مربوط به تکرار آپکدها در قسمت‌های مختلف برنامه، هیستوگرام‌ها با استفاده از معیار فاصله‌ی مینی‌کاوسکی که یک معیار فاصله‌ی مبتنی بر هیستوگرام‌ها است مقایسه می‌شوند و در نهایت برنامه‌های بازبسته‌بندی شده مشخص می‌شوند. جرومه و همکاران [۳۸] در پژوهش خود با استفاده از آپکدها و تکرار آن‌ها و روش‌های مبتنی بر یادگیری ماشین برنامه‌های بازبسته‌بندی شده را تشخیص می‌دهند. در پژوهشی که توسط [۳۹] و همکاران، ارائه شده است، از نمونه‌برداری مبتنی بر n-gram در ۴ اندازه‌ی متفاوت ۱ تا ۴ استفاده شده است.

^{۱۱} MetaData

^{۱۲} Sliding Window

^{۱۳} Edit Distance

^{۱۴} Manifest

برای شباهت‌سنجی از روش‌های طبقه‌بندی مبتنی بر درخت‌تصمیم، شبکه‌های عصبی و بردار ماشین استفاده شده‌است.

آقای لین و همکاران [۴۰] در این پژوهش، با استفاده از فراخوانی‌های سیستمی صدا زده‌شده توسط برنامه، رفتار آن را طبقه‌بندی می‌کنند. به عقیده‌ی این پژوهش، از آنجایی که اکثر بدافزارهای هم‌خانواده، در بازبسته‌بندی برنامه‌های اندرویدی، رفتار مشابه یکدیگر دارند، بنابراین استفاده از فراخوانی‌های سیستم و اسخران آن‌ها از سطح دالویک بایت‌کدها و سطح نخ، می‌تواند امضاء یکتایی از هر برنامه تولید کند. پس از استخراج بردار ویژگی از فراخوانی‌های موجود با استفاده از آپکدهایر برنامه، از یک طبقه‌بندی 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 [۴۱] و همکاران در پژوهش خود یک راه حل مبتنی بر استفاده از بلاک‌های ۶۴ بیتی روی فایل‌های دودویی برنامه‌های اندرویدی ارائه کرده‌اند. در روش ارائه‌شده پس از استخراج بلاک‌های ۶۴ بیتی از فایل‌ها و با استفاده از چکیده خلاصه تشابه و استخراج آنتروپی برای هر بلاک، بلاک‌هایی که کوچکتر و بزرگتر از یک حد کمینه و آستانه باشند حذف می‌شوند. سپس به هر بلوک با توجه به آنتروپی آن، یک اولویت 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 را کاهش یافته و از یک روش مبتنی بر بلوم‌فیلتر برای مقایسه و شباهت‌سنجی استفاده می‌شود.

آقای کو و همکاران [۴۲] از یک راه‌حل مبتنی بر استفاده از k-gram برای تشخیص بسته‌های بازبسته‌بندی شده استفاده کرده‌اند. در این پژوهش، از حذف عملوندهای موجود در کدهای دودویی، به جهت کاهش مثبت‌های غلط در تشخیص بسته‌های بازبسته‌بندی شده استفاده شده‌است. در پژوهشی [۴۳]، کیشو و همکاران از یک راه‌حل مبتنی بر ترکیبی از دستورات کلاسی و متدهای برنامه 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 شده انجام شده‌است.

راهول و همکاران [۴۴]، روشی را پیشنهاد کرده‌اند که در آن استخراج ویژگی مبتنی بر درخت نحو انتزاع انجام می‌شود. پژوهش پس از دستیابی به کد میانی برنامه‌های اندرویدی و تبدیل آن به مجموعه‌ای از قوانین نحوی، که به صورت مجموعه‌ای از عبارات منظم هستند، درخت سطح انتزاع را در سطح تابع

تشکیل می‌دهد و سه ویژگی تعداد ورودی هر تابع، نام توابع صدازده‌شده به صورت مستقیم و مجازی و متغیرهای شرطی را استخراج می‌شود. سپس برای طبقه‌بندی از الگوریتم نزدیک‌ترین همسایه برای 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 B+ و پایگاه داده‌ی MySQL استفاده شده‌است.

به صورت کلی می‌توان گفت که روش‌های مبتنی بر دستورات، خصوصاً روش‌هایی که به صورت مستقیم از آپکد برای تشخیص برنامه‌های بازبسته‌بنده شده استفاده می‌کنند، توانایی بالایی را در تشخیص برنامه‌های بازبسته‌بندی شده ارائه نمی‌دهند. این روش‌های 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 برنامه اصلی تعریف کرده‌اند که به نظر و با توجه به وجود مبهم‌نگارهای امروزه، این فرضی غلط و غیر قابل اتکا است.

روش‌های مبتنی بر گراف

در پژوهشی که توسط آقای کروسل [۴۵] و همکاران، ابزاری مبتنی بر گراف وابستگی توسعه داده شده‌است. در این ابزار ابتدا، برنامه‌های موجود در مخزن با استفاده از یک ابزار شباهت‌سنجی در سطح فراداده‌ی برنامه، به جهت افزایش سرعت، کاهش می‌یابد. پس از حذف کدهای کتابخانه‌ای به روش لیست سفید، امضاء هر برنامه با استفاده از گراف وابستگی استخراج شده تشکیل می‌گردد. گراف وابستگی توابع، وابستگی اجزای یک تابع از دو منظر کنترلی و داده‌ای معرفی می‌کند. وابستگی کنترلی در این پژوهش، الزام اجرای یک دستور خاص پیش از دستور دیگری است و وابستگی داده‌ای، الزام مقداردی متغیر پیش از اجرای دستور مرتبط با آن است. در قسمت شباهت‌سنجی پس از ساخت گراف وابستگی داده‌ای، با استفاده از الوریتم گراف‌های هم‌ریخت VF2 شباهت‌سنجی انجام شده و بسته‌های بازبسته‌بندی شده مشخص می‌شوند. در پژوهش دیگری که توسط آقای سان [۸] انجام شده‌است، هدف پژوهش افزایش دقت تشخیص برنامه‌های بازبسته‌بندی شده با تاکید بر شبیه‌سازی رفتار برنامه بوده‌است. در این پژوهش، واسط‌های برنامه‌نویسی برنامه‌های اندرویدی مشخص کننده رفتار اصلی برنامه در نظر گرفته شده‌است. برای ساخت گراف هر برنامه، از گراف جریان مبتنی بر فراخوانی واسط‌های اندرویدی استفاده شده و در نهایت پس از استخراج گراف، هر گراف نمایانگر امضاء یک برنامه می‌باشد. در گراف حاصل هر نود گراف حاوی اطلاعات یک واسط و یال‌های گراف شامل جریان کنترلی بین واسط‌های اندرویدی است. برای شباهت‌سنجی، از الگوریتم VF2 برای تشخیص گراف‌های هم‌ریخت استفاده شده‌است. در مرحله‌ی آخر برنامه‌هایی که امضاء مشابه و یکسانی در هم‌ریختی دریافت کرده‌اند به عنوان برنامه‌های بازبسته‌بنده شده در نظر گرفته

می‌شوند.

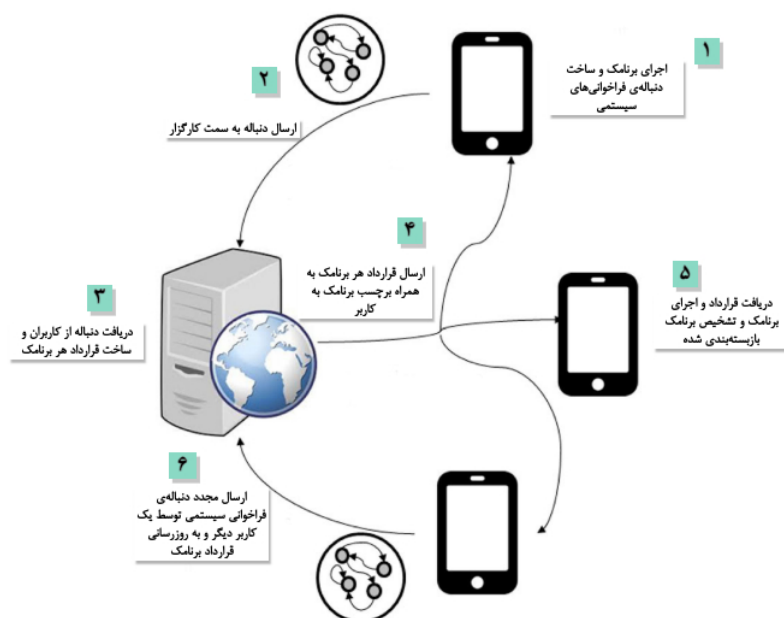
پژوهش آقای هو و همکاران [۴۶] شامل دو مرحله‌ی ساخت گراف فراخوانی متدهای برنامه و ماژول تشخیصی بازبسته‌بندی است. پس از دیس‌اسمیل کردن فایل‌های برنامه، گراف فراخوانی متدهای برنامه تشکیل شده و تشکیل جنگلی از گراف‌های متصل و جدا از هم می‌دهند. سپس با استفاده از فراخوانی واسط‌های اندرویدی موجود در هر متد، گراف به دو بخش فراخوانی‌های حساس و غیر حساس تقسیم می‌شود و با توجه به میزان حساسیت واسط‌های فراخوانی شده، امتیاز اولویت 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 برنامه‌های بازبسته‌بندی عمدتاً از روش‌های تشخیص گراف‌های هم‌ریخت استفاده می‌کنند، ژو و همکاران [۴۷] روشی برای افزایش سرعت در تشخیص ارائه کرده‌اند. در این پژوهش در ابتدا ماژول‌های اصلی برنامه که رفتار اصلی برنامه را تشکیل می‌دهند شناسایی می‌شود. برای شناسایی ماژول‌های اصلی برنامه، از یک گراف جهت‌دار مبتنی بر ارتباط بسته‌های برنامه 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 دو بسته از یک مقدار آستانه بیشتر باشد، با یکدیگر ادغام می‌شوند و رویه‌ی ادغام بسته‌ها در یک روند بازگشتی تکرار می‌شود تا زمانی که هیچ بسته‌ای را نتوان با یکدیگر ادغام کرد. در این حالت بسته‌ی نهایی شامل بسته‌ی اصلی برنامه است که رفتار برنامه بر اساس آن مشخص می‌شود. 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 شده است. برای مقایسه‌ی میان ماژول‌های اصلی برنامه، ابتدا اصلی‌ترین ماژول شامل بیشترین تعداد فعالیت، مشخص می‌شود و مقایسه میان ماژول‌های اصلی برنامه‌های اندرویدی، با استفاده از یک بردار ویژگی متشکل از فراخوانی واسط‌ها و مجوزهای درخواستی انجام می‌شود. برای مقایسه از درخت VP استفاده شده است که منجر به افزایش سرعت در کنار دقت تشخیص مناسب شده است. برخلاف روش‌های معمول گرافی و در نهایت مقایسه‌ی دودویی، روش پیاده‌سازی شده در پژوهش ژو، با مرتبه‌ی زمانی $n \lg n$ یکی از پر سرعت‌ترین روش‌های مبتنی بر تشکیل گراف می‌باشد.

از آنجایی که تشخیص برنامه‌های بازبسته‌بندی شده به وسیله‌ی مقایسه‌ی گرافی، نیازمند الگوریتم‌های تشخیص گراف‌های هم‌ریخت است که معمولاً سرعت بسیار پایینی دارند، چن و همکاران [۴۸] با استفاده از مدل کردن گراف به یک فضای سه‌بعدی، سرعت تشخیص و مقایسه‌ی گراف‌های هم‌ریخت را به مراتب افزایش داده‌اند. ایجاد کدهای مرده در کدهای برنامه، منجر به تغییر گراف جریان برنامه‌های اندرویدی می‌شود به همین جهت در این پژوهش تمامی گره‌های گرافی که نشان‌دهنده‌ی متدهای برنامه 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 کاملاً متناظر، از مقایسه‌ی فاصله‌ی گره‌های متناظر استفاده می‌شود. روش ارائه‌شده علاوه بر مقاومت بالا مقابل مبهم‌نگاری، ناشی از مدل‌کردن برنامه‌ها در یک فضای گرافی، به دلیل استفاده از روشی نو در مقایسه‌ی گراف‌های مخزن برنامه‌ها، سرعت بسیار بالاتری از روش‌های پیشین دارد. برای حذف کتابخانه‌های اندرویدی از روش لیست سفید مبتنی بر اندازه‌ی بسته‌های مورد مقایسه استفاده شده‌است.

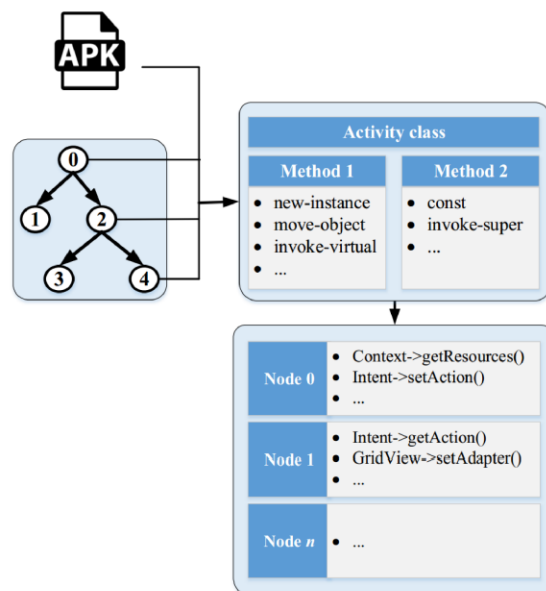
بر خلاف پژوهش‌های رایج در حوزه‌ی تشخیص بازبسته‌بندی برنامه‌های اندرویدی، در پژوهش آقای آلدینی و همکاران [۴۹]، از یک معماری کارخواه و کارگزار برای تشخیص برنامه‌های اندرویدی بازبسته‌بندی شده استفاده شده‌است. در این معماری یک برنامه‌ها بر روی دستگاه اندرویدی کاربران نصب می‌شود و شروع به ثبت و ارسال فراخوانی‌های سیستمی به کارگزار می‌کند.



شکل ۳-۱: مراحل تشخیص برنامه‌های بازبسته‌بندی شده در پژوهش آقای آلدینی

ایده‌ی پژوهش، استفاده از فراخوانی‌های سیستمی برای شبیه‌سازی ایستای رفتار برنامه‌های اندرویدی بوده است. اثرانگشت برنامه‌ها توسط گراف فراخوانی‌های سیستمی ارسالی از سمت کاربران در سمت کارگزار، انجام می‌شود سپس با استفاده از اثرانگشت موجود، یک مدل برنامه‌ها تحت عنوان قرارداد ساخته شده و این مدل به برنامه‌ی کارخواه فرستاده می‌شود. در سمت کارخواه، برنامه‌ها نصب‌شده در مرحله‌ی اول، فراخوانی‌های سیستمی برنامه‌ها موجود را با قراردادهای فرستاده‌شده مطابقت می‌دهد و در صورتی که فاکتورهایی نظیر نوع و تعداد فراخوانی‌های اندرویدی برنامه‌ها با مدل پیش‌بینی شده یکسان نباشد، هشدار لازم از طریق برنامه‌ها کارخواه به کاربر داده می‌شود. در کنار استفاده از گراف فراخوانی‌های سیستمی،

پیاده‌سازی یک معماری کارخواه و کارگزار یکی از ایده‌های اولیه‌ی این پژوهش بوده‌است. همچنین به دلیل استفاده از این معماری، پردازش سمت کارخواه به حداقل خود رسیده است و تحلیل برنامه نیازمند هیچ دانش اولیه‌ای از سمت کارخواه نمی‌باشد. علاوه بر این، تعداد برنامه‌های موجود در مخزن کارگزار، به صورت مرتب افزایش پیدا کرده و این موجب پویایی مخزن برنامه‌های اندرویدی پژوهش می‌شود. پژوهش دیگری در زمینه‌ی روش‌های تشخیص بازسته‌بندی مبتنی بر گراف توسط جنگ و همکاران [۵۰] ارائه شده‌است. در این پژوهش، پس از استخراج گراف ارتباط بین کلاس‌ها از داخل دالویک بایت‌کدهای برنامه اندرویدی، امضای هر برنامه شامل یک بردار ویژه، متشکل از فراخوانی‌های کلاسی برنامه می‌باشد. مقایسه‌ی بردارهای ویژه‌ی برنامه‌های موجود به وسیله‌ی الگوریتم یافتن بزرگ‌ترین زیردنباله‌ی مشترک انجام شده و تعیین یک حد آستانه در زیردنباله‌های مشترک، برنامه بازسته‌بندی شده شناسایی می‌شود. روش مورد نظر را می‌توان به نوعی یک روش مبتنی بر گراف و دنباله‌ی آپکدهای برنامه 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 [۳] مطرح شده‌است. پژوهش مورد نظر مبتنی بر استخراج گراف فعالیت در برنامه‌های اندرویدی است. گره‌های گراف مورد نظر شامل لیستی از واسطه‌های فراخوانی شده در آن فعالیت و یال‌های گراف، نشان‌دهنده‌ی یک انتقال از یک فعالیت به فعالیت دیگر است. یکی از معایب این پژوهش استفاده از الگوریتم VF۲ به عنوان الگوریتم اصلی برای مقایسه و یافتن گراف‌های هم‌ریخت است که باعث کاهش سرعت پژوهش می‌شود.



شکل ۳-۲: هر گره از گراف در پژوهش نگویان شامل لیستی از واسطه‌های فراخوانی شده در آن فعالیت است. [۳]

به عنوان آخرین پژوهش صورت گرفته در این قسمت، پژوهش [۵۱، ۵۲] را بررسی خواهیم کرد.

پژوهش [۵۲] که الهام‌دهنده‌ی پژوهش [۵۱] می‌باشد با استفاده از پیمایش گرافی بر روی گراف جریان برنامه‌های اندرویدی و استخراج ویژگی‌های متعدد نظیر واسطه‌های اندرویدی، فراخوانی توابع و فیلترهای ساختاری نظیر اندازه‌ی طول امضا، امضای هر کلاس را تشکیل می‌دهد و امضاء هر برنامه حاصل از الحاق تمامی کلاس‌های برنامه (مرتب شده به صورت الفبایی) می‌باشد. تفاوت این دو پژوهش بیشتر در ساختار امضاء هر برنامه می‌باشد که به تفصیل در فصل ۴ به تفصیل بیشتر این دو پژوهش و شرح تفاوت‌های آن خواهیم پرداخت.

به طور کلی می‌توان گفت که روش‌های گرافی تشخیص برنامه‌های بازبسته‌بندی شده از دقت بالایی در تشخیص برخوردار هستند اما اکثر روش‌های ارائه‌شده در این دسته، خصوصاً آن‌هایی که در نهایت هر برنامه 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 مدل‌ها گرافی را در فضای دیگری بررسی کنند، سرعت تشخیص پژوهش بالاتر رفته و می‌توان از دقت در تشخیص مسایل گرافی نیز استفاده نمود.

روش‌های مبتنی بر تحلیل ترافیک شبکه

در این قسمت به بررسی پژوهش‌های صورت‌گرفته در حوزه‌ی تشخیص برنامه‌های بازبسته‌بندی شده مبتنی بر تحلیل ترافیک شبکه می‌پردازیم. به عنوان اولین پژوهش مورد بررسی، پژوهش ارائه‌شده توسط وو [۵۳] و همکاران را بررسی خواهیم کرد. در این پژوهش امضای هر برنامه مبتنی بر ترافیک http تولیدشده توسط آن درست می‌شود. در مرحله‌ی اول این پژوهش، تمامی ترافیک تولیدشده توسط برنامه جمع‌آوری شده و در مرحله‌ی بعدی ترافیک http جداسازی و تحلیل روی این دسته ادامه پیدا می‌کند. ترافیک حاصل از برنامه‌های اندرویدی در این پژوهش به دو دسته‌ی کلی تقسیم می‌شود:

- ترافیک مرجع: ترافیک تولیدشده توسط برنامه توسعه‌یافته

- ترافیک کتابخانه‌ای: ترافیک تولیدشده توسط کدهای کتابخانه‌ای

جهت جداسازی ترافیک مرجع و ترافیک کدهای کتابخانه‌ای از الگوریتم‌های تطبیق جریان ترافیک http و الگوریتم تطبیق هانگ‌رین استفاده شده‌است. در قسمت شباهت‌سنجی از درخت جست‌وجوی VPT به جهت افزایش سرعت از جهت توازن درخت، استفاده شده‌است. پژوهش مورد نظر ایده‌ای نو در زمینه‌ی تشخیص برنامه‌های بازبسته‌بندی شده است اما مشکل اصلی این پژوهش آن است که در مقابل ترافیک رمز نشده مقاوم نیست و نمی‌توان از این روش استفاده کرد. در پژوهشی که توسط الشهری [۵۴] ارائه‌شده

است تفکیک ترافیک به وسیله‌ی یک طبقه‌بند انجام می‌شود. پس از تفکیک ترافیک متغیرهای هر بسته شامل Request, Value, Get, Host مشخص می‌شود. ترافیک‌های از نوع Request در این قسمت به دو نوع اجباری و یا غیراجباری تقسیم می‌شود. جداسازی ترافیک‌های شبکه از آن جهت اهمیت دارد که بسیاری از کتابخانه‌های رایگان و یا حتی بدافزارهای موجود، یک نقطه‌ی دسترسی توسط واسطه‌های برنامه‌نویسی برای دسترسی به کتابخانه‌ها ایجاد کرده‌اند که برنامه‌های اندرویدی به وفور از این درگاه‌ها استفاده می‌کنند به جهت جلوگیری از رخداد منفی غلط زیاد در تشخیص جداسازی ترافیک اجباری از غیر اجباری، موجب افزایش دقت تشخیص می‌شود. برای شباهت‌سنجی ترافیک اجباری شامل ترافیک اصلی برنامه، از یک الگوریتم فاصله - درخواست که مبتنی بر فاصله‌ی اقلیدوسی است مورد استفاده قرار گرفته‌است. پیشنهاد جداسازی ترافیک کتابخانه‌ای و اصلی به وسیله‌ی نقطه‌های دسترسی عمومی 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 [۵۵] ارائه شده، از یک طبقه‌بند به جهت تشخیص برنامه‌های 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 کرده و پس از استخراج ویژگی‌هایی نظیر محتوای اطلاعات هر بسته آن‌ها را برای محاسبه سمت یک کارگزار مرکزی که به صورت ابری 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 بر روی ترافیک رمزگذاری نشده (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 هر جریان به جهت کاهش اندازه، مانند پاسخ درخواست‌ها و قسمتی از سربرگ ترافیک، طبقه‌بندی صورت می‌گیرد. برای طبقه‌بندی از الگوریتم پرسرعت پژوهش [۵۶] استفاده شده‌است. مالک و همکاران [۵۷] در پژوهش دیگری تحت عنوان CREDROID از یک روش مبتنی بر درخواست نام دامنه برای تشخیص برنامه‌های 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 جداسازی و برای هر کاربر برچسب‌گذاری می‌شود. در قسمت شباهت‌سنجی ترافیک درخواستی از یک روش مبتنی بر فاصله‌ی اقلیدوسی برای بررسی متن درخواست‌های دامنه استفاده و در نهایت برنامه‌های بازبسته‌بندی شده شناسایی می‌شوند.

از آنجایی که بازبسته‌بندی برنامه‌های اندرویدی روشی محبوب برای حمله‌کنندگان به جهت تزریق گسترش بدافزارهای اندرویدی است، تمرکز پژوهش [۵۸] و همکاران بر روی برنامه‌های اندرویدی بازبسته‌بندی شده و حاوی ترافیک مشکوک به بدافزار، با توجه رفتار درخواست دامنه می‌باشد. این روش، مبتنی بر

درخواست نام دامنه توسط برنامه‌های بازبسته‌بندی شده و سپس ارتباط با آدرس به دست آمده از درخواست بوده است. سپس ترافیک فرستاده شده از سمت کاربر برای آدرس‌هایی که به دست آمده‌اند بازبینی شده و جمع‌آوری می‌شود و سپس با استفاده از روشی مبتنی بر تطابق رشته‌های درخواست، شباهت‌سنجی صورت گرفته و برنامه‌های بازبسته‌بندی شده مشخص می‌شوند. یکی از ایرادات این پژوهش آن است که مبتنی بر رفتار بدافزارهای بیشتر شناخته شده انجام شده است و در صورتی که بدافزاری رفتار مشابه با بدافزارهای محبوب نداشته باشد شناسایی نمی‌شود. همچنین تحلیل ترافیک شبکه‌ی کاربر در ترافیک خام و رمزگذاری نشده صورت می‌گیرد و در صورتی که بدافزار بازبسته‌بندی شده حاوی ترافیک رمزگذاری شده 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 [۵۹] استفاده شده است. تشخیص برنامه‌های بازبسته‌بندی شده در این پژوهش مبتنی بر طبقه‌بندی برنامه‌ها با استفاده از تحلیل ترافیک شبکه و بررسی مجوزهای دسترسی مورد نیاز برنامه می‌باشد. پس از دیس‌اسمبل برنامه، مجوزهای دسترسی از فایل فراداده‌ی برنامه مورد نظر استخراج شده و تشکیل یک بردار ویژگی دودویی را می‌دهند. این پژوهش از دو سطح برنامه‌های مورد نظر را بررسی می‌کند، در سطح اول اگر بردار ویژگی برنامه با برداری از برنامه‌های بازبسته‌بندی شده تطابق داشته باشد، آن‌گاه برنامه مورد نظر به عنوان یک برنامه مشکوک به سطح بعد فرستاده می‌شود. در سطح دوم، ترافیک رمزگذاری نشده‌ی برنامه تفکیک شده و تحلیل روی ترافیک TCP ادامه می‌یابد، سپس با استفاده از یک طبقه‌بند درخت تصمیم، برنامه‌های بازبسته‌بندی شده با استفاده از تحلیل ترافیک کاربران، مشخص می‌شوند.

روش‌های مبتنی بر تحلیل منابع

عمده‌ی پژوهش‌های موجود در این دسته، فرض تغییر در منابع برنامه‌های بازبسته‌بندی شده را مبنای تعریف خود قرار داده‌اند. در این دسته از تعریف بازبسته‌بندی، متقلب با تغییر منابع برنامه که منجر به تغییر رابط کاربری آن خواهد شد، سعی در جعل برنامه اصلی دارد. در این حالت رابط کاربری برنامه مشابه با رابط کاربری برنامه اصلی است اما کارکرد آن دچار تغییرات زیادی می‌شود بنابراین شناسایی این دسته از برنامه‌ها با استفاده از ویژگی‌های مبتنی بر منابع آن انجام می‌شود. به عنوان اولین پژوهش از این دسته، شائو و همکاران [۱]، برای شکل امضا هر برنامه، از دو دسته ویژگی آماری و ساختاری استفاده می‌کنند. در قسمت ویژگی‌های آماری، ۱۵ ویژگی شاخص از منابع برنامه‌های اندرویدی نظیر تعداد فعالیت‌ها، تعداد مجوزهای دسترسی، تعداد فیلترهای هدفمند و میانگین تعداد فایل‌های png و xml استخراج می‌شود. علاوه بر این ۵ ویژگی، ۱۰ ویژگی دیگر آماری شامل میانگین فراخوانی ۱۰ منبع پر بازدید

در برنامه‌های اندرویدی، مطابق با جدول ۳-۱، استخراج می‌شود.

جدول ۳-۱: 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 |

پس از استخراج ۱۵ ویژگی اماری، دو ویژگی ساختاری دیگر مبتنی بر لایه‌ی فعالیت و کنترل‌گر رویدادهای برنامه‌ساخته می‌شود. برای ساخت این دو ویژگی از یک درخت انتزاعی مبتنی بر فعالیت‌های برنامه، استفاده شده‌است. در قسمت شباهت‌سنجی، پس از نرمالایز کردن ویژگی‌های استخراج شده در مراحل قبل، از دو روش طبقه‌بندی مبتنی بر نزدیک‌ترین همسایه و طبقه‌بندی طیفی برای تشخیص برنامه‌های تقلبی استفاده شده‌است.

در پژوهش دیگری که توسط لین و همکاران [۶۰] انجام شده‌است، تشخیص بازبسته‌بندی با استفاده از چکیده‌ی تصاویر موجود در فایل‌های APK و 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 برنامه‌هایی که امتیاز تشابه آن‌ها از یک حد آستانه بیشتر باشد به عنوان برنامه‌های بازبسته‌بندی شده تشخیص داده می‌شوند. یکی از مفروضات پژوهش، به جهت کاهش فضای مقایسه، عدم لورفتن کلید خصوصی توسعه‌دهنده‌ی برنامه می‌باشد. علاوه بر این، استفاده از روش‌های چکیده‌سازی معمول در این پژوهش، منجر به مقاومت پایین آن در مقابل مبهم‌نگاری می‌شود. سان و همکاران [۶۱] با استفاده از تحلیل گراف فعالیت‌های برنامه، مجموعه‌ای از تصاویر هر برنامه را تحت هر فعالیت استخراج کرده و هر دید 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 برنامه، گرافی تخت عنوان گراف گروه-ناحیه ساخته می‌شود. برای مقایسه و شباهت‌سنجی، با استفاده از پیمایش گراف‌ها، گره‌های

گراف به عنوان نواحی برنامه 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 [۶۲] با بررسی برنامه‌های بازبسته‌بندی شده متوجه شدند که ویژگی‌های ساختاری واسطه‌های کاربری 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 واسطه‌های کاربری را استخراج و تحت عنوان امضای برنامه استفاده می‌کنند و در ادامه این سه ویژگی را به یک فعالیت نسبت می‌دهند. در نهایت با کنار هم قرار گرفتن سه تایی‌های هر فعالیت و مقایسه‌ی آن‌ها با یکدیگر برنامه‌های تقلبی مشخص می‌شوند. هدف آقای لین در پژوهش [۶۳] کاهش نرخ منفی غلط و مثبت غلط بوده‌است. در این پژوهش از ساختار سلسله‌مراتبی فایل‌های موجود در یک apk استفاده شده‌است. ساختار سلسله‌مراتبی مبتنی بر فایل‌های برنامه 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 مثبت غلط به شدت افزایش پیدا می‌کند. برای حل این مشکل نویسنده، از یک ساختار چند مرحله‌ای برای تشخیص برنامه‌های 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 می‌تواند به راحتی با ایجاد تعداد زیادی فایل بلااستفاده، سیستم تشخیص را دور بزند. گوآ و همکاران [۶۴] با استفاده از طرح‌بندی و ایجاد درخت برنامه‌های بازبسته‌بندی شده را تشخیص می‌دهند. ابتدا درختی از تمامی مولفه‌های از نوع منبع و با استفاده از فایل‌های xml به نام درخت طرح‌بندی کامل، استخراج می‌شود. درخت مذکور حاوی تمامی مولفه‌های طرح‌بندی یک برنامه 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 این دسته محبوبیت زیادی دارد، برای مثال، در روش [۶۵]، ابتدا طرح‌بندی‌های مربوط به کتابخانه‌های اندرویدی حذف شده، سپس هر طرح‌بندی به یک درخت نگاشت می‌شود. به جهت مقایسه، طرح‌بندی‌های برنامه با یکدیگر ادغام می‌شوند و درخت نهایی در قابل یک فایل XML کدگذاری می‌شود و در ادامه با استفاده از چکیده‌سازی فازی میان فایل‌های XML برنامه‌های بازبسته‌بندی شده مشخص می‌شوند. در نمونه‌ی دیگری، لیو و همکاران [۶۶] در یک ساختار دو مرحله‌ای

مبتنی بر طرح‌بندی برنامه‌های بازبسته‌بندی شده را تشخیص می‌دهند. مرحله‌ی اول در این پژوهش شامل تشخیص برنامه‌های مشکوک به صورت درشت‌دانه است که با استفاده از تصاویر موجود در پوشه‌ی منابع انجام می‌گیرد. در ادامه و به صورت ریزدانه، درخت طرح‌بندی تشکیل شده و تشکیل امضای هر برنامه را 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 [۶۷، ۶۸] ما و همکاران [۶۸] با معرفی گراف انتزاعی طرح‌بندی و انتقالی، ALTG برنامه‌های بازبسته‌بندی شده را شناسایی می‌کنند. در این پژوهش در ابتدا طرح‌بندی‌های هر برنامه تحت عنوان یک گراف جهت‌دار مدل می‌شود که هر گره گراف مذکور شامل طرح‌بندی‌های مشابه می‌باشد. در مقایسه با [۶۷]، روش ما و همکاران، 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 تطبیف گراف پیشنهاد شده که ارزیابی عملکرد پژوهش نسبت به دو پژوهش مورد مقایسه، افزایش سرعت و دقت را به همراه داشته است.

۳-۲-۳ مبتنی بر تحلیل پویا

به صورت کلی پژوهش‌های صورت‌گرفته در این قسمت را می‌توان به دو بخش روش‌های مبتنی بر جعبه‌شن و یا روش‌های خودکار تقسیم‌بندی کرد. در روش‌های مبتنی بر جعبه‌شن، پژوهش‌کنندگان با ایجاد محیطی شبیه‌سازی شده و تعامل حداکثری با برنامه اندوریدی مدنظر سعی در شبیه‌سازی رفتار برنامه مورد نظر و جمع‌آوری مجموعه‌ای از ویژگی‌های مدنظر 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 برنامه به طوری که تمامی رفتارهای آن‌را شبیه‌سازی کند، نیازمند زمان زیادی است که ممکن است روش را ناکارآمد کرده و بررسی برنامه‌های موجود را محدود نماید. علاوه بر این ایجاد محیطی که بتواند به صورتی کامل رفتار برنامه را شبیه‌سازی کند نیازمند آن‌است که به نوعی تمامی خدمات هر برنامه منحصرًا بررسی شود که این عملی تقریباً ناممکن است مگر آن‌که شبیه‌سازی منحصر یک برنامه خاص پیاده‌سازی شود. ایراد دوم آن است که برنامه‌های بازبسته‌بندی شده که دارای بدافزار هستند، ممکن است پیاده‌سازی آن‌ها قادر به شناسایی محیط شبیه‌سازی شده باشد و در نتیجه، از اجرای قسمت‌هایی از برنامه 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 برنامه‌های اندرویدی دارند. بیشتر روش‌ها در این دسته از معماری‌های ابری و کارخواه-کارگزار برای شناسایی برنامه‌های بازبسته‌بندی شده با استفاده از ارسال گزارشات از سمت کاربران استفاده می‌کنند. چندین نمونه از این روش‌ها را مانند [۴۹، ۴۰] در قسمت‌های پیشین بررسی کردیم. روش‌های مبتنی بر تحلیل خودکار هیچ شبیه‌سازی در مورد برنامه انجام نمی‌دهند

و رفتار برنامه را با توجه به رفتار کاربران مورد تحلیل قرار می‌دهند.

در پژوهش والرئو و همکاران [۶۹] با استفاده از اجرای برنامه‌های اندرویدی در چندین جعبه‌شن مشهور و جمع‌آوری اطلاعات برای هر برنامه اندرویدی، یک پروفایل - استفاده ساخته می‌شود. این پروفایل نشان‌دهنده‌ی استفاده‌ی برنامه از ۸ ویژگی مبتنی بر منابع سیستمی شامل لیست تماس‌ها، موقعیت مکانی، پیامک، شبکه‌ی ارتباطی بی‌سیم، برنامه‌ها، باتری و تماس‌ها می‌باشد. روند کل جمع‌آوری امضای برنامه به این شکل است که برنامه‌های موجود در مخزن توسط یک مولفه در چندین جعبه‌شن مشهور و در دسترس اجرا شده و ویژگی‌های ذکر شده استخراج می‌شود و در نهایت شکل یک بردار ویژگی می‌دهند. در مرحله‌ی انتهایی، بردارهای ویژگی برنامه‌های اندرویدی با یکدیگر مقایسه شده و بسته‌های بازبسته‌بندی شده مشخص خواهند شد.

از آنجا که بررسی پردازش برنامه‌های اندرویدی به ازای ورودی‌های یکسان می‌تواند نتایج مشابه در برنامه‌های اندرویدی داشته باشد، ژانگ و همکاران [۷۰] روشی را مبتنی بر زمان مورد نیاز برای اجرای ورودی‌های یکسان توسط پردازنده پیاده‌سازی کرده‌اند. در این پژوهش، ابتدا تعدادی ورودی به هر برنامه اندرویدی موجود در مخزن داده شده و پس از اتمام پردازش، بازه‌ی زمانی اجرای پردازش در پردازنده ضبط می‌شود و تحت عنوان یک دوتایی <پردازش، زمان> نمایش داده می‌شود. ورودی‌های یکسان برای تمامی برنامه‌های مورد بررسی اجرا می‌شود و در نهایت لیستی از دوتایی‌های موجود برای هر برنامه، شکل امضای آن را می‌دهد.

در روش دیگری که توسط نگویان و همکاران [۷۱] اجرا شده، برنامه‌های اندرویدی با یک روش مبتنی بر جعبه‌شن و استخراج ویژگی‌هایی مبتنی بر واسطه‌کاری برنامه اندرویدی، بسته‌های بازبسته‌بندی شده را شناسایی می‌کنند. در این پژوهش با استفاده از یک جعبه‌شن که وظیفه‌ی آن ضبط تصاویر محیط کاربر برنامه می‌باشد، تصاویری از برنامه‌های اندرویدی مورد تست ذخیره می‌شود. از آنجایی که روش‌های چکیده‌سازی مرسوم، چکیده‌ای کاملاً متفاوت حتی برای کوچکترین تغییرات تولید می‌کنند. در این پژوهش از روشی مبتنی بر چکیده‌سازی ادراکی استفاده شده است. در این دسته از روش‌های چکیده‌سازی، تغییرات کوچک به همان اندازه باعث تغییرات کوچک از چکیده‌ی تولیدشده می‌شود و فایل‌های مشابه اما با تغییرات اندک می‌توانند از مقایسه‌ی چکیده‌سازی ادراکی، شناسایی شوند. در مرحله‌ی شباهت‌سنجی، تصاویر ضبط‌شده از هر برنامه، با استفاده از روش‌های میانگین چکیده‌سازی، با تصاویر برنامه‌های موجود در مخزن مورد مقایسه قرار می‌گیرد و در نهایت برنامه‌های بازبسته‌بندی شده به این طریق تشخیص داده می‌شوند. در پژوهش دیگری که مبتنی بر واسطه‌های کاربری انجام شده است، سو و همکاران [۷۲] با استفاده از استخراج فعالیت‌های برنامه از فایل‌های فراده‌ی 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 برنامه‌ها، استخراج می‌شود و تشکیل بردار ویژگی

هر برنامه را می‌دهند. در ادامه با استفاده از روش‌های مبتنی بر چکیده‌سازی محلی، شباهت بردارهای ویژگی با یکدیگر سنجیده و در صورتی که میزان شباهت از یک حد آستانه بیشتر باشد، جفت مورد بررسی، بازبسته‌بندی شناسایی می‌شوند. برنامه برای مقاومت بیشتر در مقابل مبهم‌نگاری از لیست فعالیت‌های برنامه استفاده کرده و نقاط ورود را از روی این لیست فعالیت مشخص می‌کند. این پژوهش، فعالیت‌های موجود در برنامه‌های 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 برنامه‌های اندرویدی، فعالیت‌های موجود کاملاً به هم وابسته هستند و گاهی یک فعالیت نتیجه‌ی یک فعالیت دیگر است بنابراین فرض پژوهش در این قسمت به نظر اشتباه می‌باشد.

به طور کلی، تشخیص برنامه‌های اندرویدی مبتنی بر روش‌های پویا شامل استخراج ویژگی‌های متنوع ساختاری و رفتار برنامه در حین اجرای آن در محیط شبیه‌سازی شده است. برای مثال، در پژوهش [۷۳] با استفاده از یک برنامه تزریق‌شده به هسته‌ی سیستم‌عامل، تمامی فراخوانی‌های سیستمی ناشی از اجرای برنامه در محیط جعبه‌شن، استخراج می‌شود. استخراج فراخوانی سیستمی از طریق هسته‌ی سیستم‌عامل، منجر به مقاومت بالای این روش مقابل مبهم‌نگاری می‌شود چرا که مبهم‌نگاری‌ها اغلب در سطح برنامه انجام می‌شود و دستورات موجود در سمت هسته‌ی سیستم‌عامل تغییر نخواهد کرد. کیم و همکاران [۷۴]، برای تشخیص برنامه‌های بازبسته‌بندی شده، از بردارهای ویژگی مبتنی بر فراخوانی‌های واسطه‌های برنامه‌نویسی استفاده کرده‌اند و در نهایت با مقایسه‌ی برنامه‌ها، مسئله را حل می‌کنند. گوان و همکاران [۷۵]، با استفاده از یک روش مبتنی بر معناشناسی در برنامه‌های اندرویدی، بسته‌های بازبسته‌بندی شده شناسایی می‌کنند. آن‌ها با بررسی مشکلات تشخیص در تحلیل‌های ایستا، در ابتدا گراف جریان ارتباطی مبنا کلاس‌های برنامه را استخراج کرده و در ادامه با استفاده از وزن‌دهی گراف، مبتنی بر ورودی و خروجی کلاس، کلاس‌های اصلی برنامه 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 دست می‌آیند. تشخیص بازبسته‌بندی با استفاده از یک حد آستانه، بر اساس تعداد متدهای یکسان مشخص می‌شود. یو و همکاران [۶۷] با استفاده از ساخت گراف فعالیت مبتنی بر واسطه کاربری هر برنامه و در نهایت مقایسه‌ی گرافی، برنامه‌های 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.

۳-۳-۳ سایر روش‌ها

وانگ و همکاران [۷۶] از روشی مبتنی بر طبقه‌بندی برای شناسایی کتابخانه‌های اندرویدی استفاده کرده‌اند. با توجه به این که کتابخانه‌های اندرویدی در برنامه‌های مختلف تکرار می‌شوند بنابراین جمع‌آوری دنباله‌ی فراخوانی واسطه‌های برنامه‌نویسی در کتابخانه‌های مشابه، منجر به ساخته‌شدن دنباله‌های مشابه می‌شود. بنابراین پس از جمع‌آوری دنباله‌ی فراخوانی‌ها از برنامه‌های متعدد، طبقه‌بندی مبتنی بر دنباله‌ی ورودی با استفاده از داده‌ی TRAIN 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 برنامه‌های اندرویدی بازبسته‌بندی شده استفاده شده‌است. پس از حذف کدهای کتابخانه‌ای، دنباله‌های فراخوانی واسطه‌های اندرویدی در هر برنامه محاسبه و دنباله‌های موجود با یکدیگر مقایسه می‌شوند. در این مرحله، برنامه‌های مشکوک به بازبسته‌بندی تشخیص داده می‌شوند و در مرحله‌ی بعدی بررسی دقیق‌تر صورت می‌گیرد. در مرحله‌ی دوم، متغیرهای موجود در برنامه در روشی مبتنی بر توکن، به صورت برداری نشان داده می‌شوند. به جهت شباهت‌سنجی و مقایسه‌ی بردارهای موجود، تعداد بردارهای یکسان، که مبتنی بر شناسه‌ی هر متغیر و نام است، با یکدیگر مقایسه می‌شود و در صورتی که تعداد توکن‌های مشابه، در یک جفت برنامه از یک حد آستانه بیشتر باشد، جفت مورد بررسی به عنوان بازبسته‌بندی در نظر گرفته می‌شوند. روش پژوهش، از آنجا که در دو مرحله برنامه‌های بازبسته‌بندی شده را شناسایی می‌کند از سرعت خوبی برخوردار است، اما با استفاده از روش‌های مبتنی بر افزودن متغیرهای، می‌توان روش یادشده را دور زد.

نیشا و همکاران [۷۷]، با استفاده از ویژگی‌های مبتنی بر مجوزهای دسترسی و ایجاد بردار بیتی مجوزهای هر برنامه، به وسیله‌ی روش‌های یادگیری ماشین برنامه‌های 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 برنامه‌های اصلی استفاده شده‌است که باعث می‌شود روش موجود در عین داشتن دقت بالا در مورد مجموعه‌ی آزمون، به صورت کلی مقاومت بالایی در تشخیص بازبسته‌بندی نداشته باشد. مشکل دیگر این روش، آن است که عملاً تعریف تشخیص بازبسته‌بندی در این روش‌ها، حل مسئله‌ی تصمیم در مورد بازبسته‌بندی است و شامل تشخیص جفت تقلبی نمی‌شود.

۴-۳ پیش‌گیری از بازبسته‌بندی

عمده‌ی روش‌های موجود به جهت پیش‌گیری از بازبسته‌بندی نیازمند ایجاد تغییراتی در برنامه پیش از انتشار آن به صورت عمومی می‌باشد. این روش‌ها، از بازبسته‌بندی برنامه توسط متقلب‌ان جلوگیری کرده و یا در صورتی که بازبسته‌بندی صورت بگیرد توسعه‌دهندگان متوجه آن خواهند شد. مزیت بزرگ روش‌های

پیش‌گیری از باز بسته‌بندی آن است که از قطعیت بیشتری برخوردار هستند، یعنی در صورتی که باز بسته‌بندی صورت بگیرد اکثراً به صورت قطعی توانایی تشخیص و جلوگیری از انتشار برنامه‌ها را دارند. صورت کلی پژوهش‌های اخیر در این حوزه را می‌توان به دو دسته تقسیم کرد که در ادامه به توضیح مختصری از هر کدام می‌پردازیم:

۳-۴-۱ روش‌های مبتنی بر نشان‌گذاری

در این دسته از روش‌ها که نیازمند ایجاد تغییراتی در کد برنامهک توسعه داده شده است، توسعه‌دهندگان قسمتی از امضای خود را تحت عنوان حق‌تکثیر در برنامه پنهان می‌سازند و در صورتی که برنامه جدیدی باز بسته‌بندی شود با توجه به تغییر این حق‌تکثیر قادر به تشخیص و جلوگیری از باز بسته‌بندی خواهند بود. به صورت کلی روش‌های نشان‌گذاری به دو نوع ایستا و پویا تقسیم می‌شوند. در روش‌های ایستا، پیش از ساخته شدن برنامهک و انتشار آن، حق‌تکثیر ایجاد و در ساختار داده‌ای و یا کدهای برنامهک تزریق می‌شود اما در روش‌های پویا، به جهت غلبه بر محدودیت روش‌های ایستا و بررسی دقیق‌تر، حق‌تکثیر توسعه‌دهندگان در حین اجرای برنامهک 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 تزریق شده به برنامهک‌ها استفاده می‌شود که عدم باز بسته‌بندی 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 برنامهک‌های باز بسته‌بندی شده می‌توانند به صورت مکمل یکدیگر عمل کنند. در این حالت ابتدا برنامهک‌های مشکوک با استفاده از روش‌های بیان شده، تشخیص داده می‌شوند و در ادامه به جهت اطمینان از صحبت برنامهک مورد نظر، توسط روش‌های مبتنی بر نشان‌گذاری، حق‌تکثیر توسعه‌دهندگان بررسی می‌شود و در صورتی که برنامهک مدنظر باز بسته‌بندی شده باشد، از فروشگاه حذف خواهد شد. استفاده از هر دوی این روش‌های سرعت و دقت تشخیص برنامهک‌های اندرویدی باز بسته‌بندی شده را افزایش می‌دهد.

یکی از اولین پژوهش‌های این حوزه توسط وو و همکاران [۷۸] انجام شده است. در این پژوهش ابتدا رشته‌ای از اعداد و حروف برای تولید نشان‌گذاری مشخص می‌شود. سپس رقم مشخص شده در فرایندی مشخص به یک گراف جایگشت نگاشت می‌شود و در ادامه گراف مورد نظر با استفاده از یک قطعه‌کد نشان‌داده می‌شود. برای جلوگیری از مبهم‌نگاری نشان در فرایند باز بسته‌بندی، کد ایجاد شده برای گراف نشان‌گذاری، در قسمت‌های مختلف برنامهک هدف تقسیم می‌شود. برای تایید برنامهک اصلی و جلوگیری از باز بسته‌بندی از یک برنامهک ثانویه که حکم فراداده‌ی برنامهک اصلی را دارد استفاده می‌شود. برای ساخت برنامهک 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 برنامه‌کد ثانویه، نگه‌داشت مکان‌هایی از برنامه‌کد 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 برنامه‌کد اصلی تقسیم می‌شوند. بنابراین در صورتی که برنامه‌کد 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 [۷۹]، از یک روش نشان‌گذاری مبتنی بر تصاویر نشان‌گذاری شده استفاده می‌کنند. نویسندگان، با توجه به این که در پژوهش [۷۸] هیچ وابستگی داده‌ای میان کدهای نشان‌گذاری شده وجود نداشته‌است، روشی را پیشنهاد داده‌اند تا مهاجم نتواند با تحلیل برنامه‌کد و یافتن کدهای نشان‌گذاری شده، مبهم‌نگاری در آن‌ها ایجاد کند و سیستم تشخیص را دور بزند. در این روش، با استفاده از کاراکترهای اسکی، تصویر ورودی را کدگذاری کرده و طی یک دنباله‌ی تصادفی از رخدادهای، کد تولیدشده‌ی اسکی در مسیر رخدادهای تولیدشده پخش می‌شود. به جهت این‌که از تغییر نشان‌ها توسط مهاجم جلوگیری شود، تعدادی از کاراکترهای تصویر ورودی را با اعداد ثابت موجود در کد برنامه‌کد جایگزین می‌سازند که این روش موجب می‌شود تصویر ورودی تغییر چندانی نکند و در عین حال وابستگی داده‌ای میان کدهای برنامه‌کد و تصویر ورودی ایجاد شود. یکی از مشکلات دو روش اخیر، عدم امکان بررسی اصالت برنامه‌کد توسط کاربران می‌باشد. برای رفع این مشکل رن و همکاران [۸۰] از کدگذاری بلوکی مبتنی بر ساختارهای خودرمزگشایی استفاده کرده‌اند. در این پژوهش، کد نشان‌گذاری شده پس از تولید، در بلوک‌های شرطی کد برنامه‌کد مورد توسعه قرار گرفته و رمزگذاری صورت می‌گیرد و در حین اجرا رمزگشایی بلوک‌های اجرایی منجر به تایید اصالت کد نشان‌گذاری 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 برنامه‌کدهای بازبسته‌بندی شده توسط خود برنامه‌کد ایده‌ای بود که در پژوهش [۸۱] توسط سان و همکاران دنبال شده‌است. در این پژوهش ابتدا تمامی ساختارهای شرطی برنامه‌کد استخراج می‌شود و با استفاده از تحلیل و تغییر شروط به صورتی که منطق برنامه‌کد تغییر نکند، کد نشان‌گذاری شده در ساختار برنامه‌کد جایگذاری می‌شود. در نهایت با استفاده از یک جریان مبتنی بر داینامیک لودینگ، در صورتی که کد نشان‌گذاری شده تغییری کرده‌باشد، از اجرای برنامه‌کد جلوگیری خواهد شد.

لوآ و همکاران [۸۲] در سال ۲۰۱۶ روشی را مبتنی بر کلید عمومی توسعه‌دهنده‌ی برنامه‌کد اندرویدی، برای جلوگیری از بازبسته‌بندی ارائه کرده‌اند. نویسندگان در طی روشی به نام شبکه‌ی پنهایی مرتبط، به عنوان ورودی کد برنامه‌کد اندرویدی به همراه کلید عمومی کاربر توسعه‌دهنده‌ی برنامه‌کد را دریافت می‌کند و با

استفاده از مقایسه‌ی کلید عمومی کدسخت شده در داخل برنامه و زیررشته‌ای از محاسبات مبهم‌نگاری شده در برنامه هنگام اجرای آن، اصالت برنامه را تشخیص می‌دهند. برای به دست آوردن کلید عمومی در حین اجرای برنامه، از ویژگی بازتاب در زبان جاوا استفاده شده است. یکی از مشکلات این پژوهش آن است که به دلیل استفاده از ویژگی بازتاب برای دستیابی به کلید عمومی، حمله‌کننده می‌تواند فراخوانی متد فراخواننده 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 برنامه را که کلید عمومی در آن کدسخت شده است را بیابد. به جهت مقابله با این نوع از حملات، زنگ در پژوهش [۸۳] راه‌حلی مبتنی بر بمب‌های منطقی ارائه کرده‌اند. بمب‌های منطقی، قطعه کدهای کوچکی هستند که تنها در حالاتی خاص فعال می‌شوند. در طی این روش، کدهای نشان‌گذاری در ساختار شرط‌های منطقی در برنامه تزریق می‌شود به صورتی که برخی از این شروط در طی اجرای برنامه در سیستم عامل کاربران معمولی اجرا می‌شود. در صورتی که مهاجم از روش‌های مبتنی بر جعبه سیاه استفاده ننماید، در آن صورت محیط اجرای وی محدود به تعدادی سیستم عامل 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 داشته باشد. با استفاده از این روش، پژوهش [۸۲] 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 فعل‌سازی شروط و در نهایت دسترسی به بلوک بمب‌های منطقی، پیش از اجرای کدهای رمزگشایی شده آن‌ها را تغییر دهد، آنگاه می‌تواند از تشخیص اصالت برنامه جلوگیری نماید. تانر و همکاران [۸۴] برای حل این مشکل از کدهای بومی استفاده کرده‌اند. از آنجایی که تغییر کدهای بومی در حین بازبسته‌بندی برای مهاجم دشوار و پرهزینه است، نویسندگان پس از دستیابی به کدهای بومی و کدهای اصلی برنامه، مجموعه‌ای از بایت‌کدهای دالویک رمزنگاری شده را در قسمت‌های مختلف برنامه اندرویدی اصلی تزریق کرده و متدهای بررسی صحت برنامه را در کدهای بومی پیاده‌سازی می‌کنند. رمزنگاری بایت‌کدها در کنار فراخوانی توابع بررسی صحت در کدهای بومی، موجب شده است که روش موجود، در مقابل تحلیل ایستا و پویا توسط مهاجم امنیت داشته باشد.

۳-۴-۲ روش‌های مبتنی بر مبهم‌نگاری

همانطور که در فصل ۲ — اشاره شد، مبهم‌نگاری یکی از روش‌های پیش‌رو برای جلوگیری از بازبسته‌بندی توسط مهاجمان است. توسعه‌دهندگان از مبهم‌نگاری استفاده می‌کنند تا تغییر دادن برنامه‌های اندرویدی خود را سخت و هزینه‌بر کنند. به صورت کلی مبهم‌نگاری در وهله‌ی اول موجب می‌شود تا ساختار منطقی

اصلی و به طور مشخص تر، بدنه‌ای اصلی منطقی برنامه‌ک از دید مهاجمان مخفی بماند. در مرحله‌ی بعدی در صورتی که مهاجم بتواند بدنه‌ی منطقی اصلی برنامه‌ک اندرویدی قربانی را متوجه شود، تغییر دادن 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 برنامه‌ک‌های اندرویدی را انجام می‌دهند. اما پژوهش‌های موجود در این قسمت نیز، به صورت پیش‌رو، روش‌هایی را توسعه داده‌اند که می‌توانند به نقش پیش‌گیری در بازبسته‌بندی برنامه‌ک‌های اندرویدی را داشته باشند. یکی از روش‌های مبهم‌نگاری در برنامه‌ک‌های اندرویدی، تغییر نام شناسه‌ها، کلاس‌ها و متدهای برنامه‌ک می‌باشد. این روش موجب می‌شود تا تحلیل و جست‌وجوی رشته‌ها در برنامه‌ک دشوار شده و مهاجم نتواند با تحلیل برنامه‌ک، آن را تغییر دهد. از آنجایی که تغییر نام در بیشتر مواقع تاثیری بر منطق برنامه‌ک ندارد، تغییر نام شناسه‌های برنامه‌ک، روشی است که ورمک و همکاران [۸۵] از آن استفاده کرده‌اند. پروگارد ابزار دیگری است که با تغییر نام کلاس‌ها، متدها و شناسه‌های موجود در برنامه‌ک 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 بیشتر از کاراکترهای اسکی انجام شود [۸۶].

برنامه‌ک‌های مبتنی بر زبان جاوا، با استفاده از پیشوندهای مشخص 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 برنامه‌ک برای مهاجم سخت شود. وانگ و همکاران [۸۷] از این روش برای افزایش مقاومت مقابل مبهم‌نگاری استفاده کرده‌اند. در روشی که توسط نویسندگان پیاده‌سازی شده است، تعدادی کلاس کمکی برای مخفی کردن کلاس‌های اصلی برنامه‌ک به بسته‌های نرم‌افزاری اضافه شده است. پژوهش [۸۵] نیز در تولید ابزار پروگارد از تغییر نام بسته‌ها استفاده کرده است.

حذف کدهای بومی‌ای که در هنگام اجرای برنامه‌ک نیازمند آن‌ها نیستیم، روش دیگری از مبهم‌نگاری برنامه‌ک‌های اندرویدی است که در پژوهش [۸۸] از آن استفاده شده است. حذف کدهای بومی موجب می‌شود که خوانایی برنامه‌ک در حین دی‌کامپایل توسط مهاجمان کاهش یابد. علاوه بر این موضوع، به جهت خوانایی پایین کدهای بومی نسبت به دالویک بایدکد که یک زبان میانی محسوب می‌شود، در پژوهش اخیر [۸۸]، نویسندگان قسمتی از برنامه‌ک را که شامل الگوریتم‌های رمزنگاری می‌شود، به صورت کدهای بومی پیاده‌سازی کرده 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 وارد ساختن بدافزار و باز بسته‌بندی است. گراف کنترل جریان نشان‌دهنده‌ی روال فراخوانی میان متدهای یک برنامه را نشان می‌دهد. باسی و همکاران [۸۹] 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، به صورت میانی، انجام می‌شود. پردا و همکاران [۹۰] 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 فراخواننده دشوار شود. افزودن کدهای بیهوده، روشی دیگری است که در پژوهش‌های [۹۱، ۹۲، ۹۳]، استفاده شده‌است. در پژوهش [۹۱]، لی و همکاران، با استفاده از افزودن کدهای nop، پرش‌های قطعی و ثبات‌های داده پردازش‌های بیهوده‌ای در برنامه اضافه می‌کنند که منطق برنامه را تغییر نمی‌دهد. راستوگی در پژوهش [۹۳]، 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 بدون شرط به ابتدای کد اصلی. ژانگ [۹۲] در پژوهش خود از این روش برای مبهم‌نگاری برنامه‌های اندرویدی استفاده کرده‌است. جابه‌جایی دستورات، روش دیگری است که در پژوهش‌های [۸۶، ۹۱، ۹۰] استفاده شده‌است. باسی و همکاران [۹۱] با استفاده از جابه‌جایی دستورات به صورت کاملاً تصادفی، با دانه‌های ناهم‌گون و در نهایت استفاده از دستوران پرش، خطوط کدهای اصلی را جابه‌جا کرده و به این شکل مهاجم را در تحلیل کدهای اصلی برنامه، نامام می‌گذارد. پردا و همکاران [۹۰] با تحلیل دالویک بایت‌کدها، بلوک‌های مستقل اجرایی در آن‌ها را استخراج کرده و ترتیب آن‌ها را عوض می‌کنند. برای تضمین اجرای منطق اصلی برنامه، از شرط‌های منطقی قطعی استفاده شده‌است.

استفاده از رمزگذاری در حین کامپایل برنامه‌های 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 برنامه تقسیم کرد. در روش‌های مبتنی بر رمزگذاری داده، قسمت‌های حساسی از کد برنامه، 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 [۸۹] با استفاده از این ایده و استفاده از رمزگذاری مبتنی بر سیستم رمز سزار، گذرواژه‌های حساس و حیاتی برنامه را رمزنگاری و در هنگام اجرا رمزگشایی می‌کند. منابع برنامه‌های اندرویدی، نیازمند کامپایل توسط ماشین مجازی جاوا نیستند و توسط کدهای برنامه، فراخوانی می‌شوند. پردا و همکاران [۹۰] 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 رمز شده از بر عهده دارد، مشخص نکردند. علاوه بر رمزگذاری منابع و اطلاعات حساس، رمزگذاری کلاس‌های برنامه نیز روش دیگری

است نویسندگان پژوهش [۹۴] 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 برنامه‌های جاوا، کلاس رمزگذاری شده را بازیابی و اجرا می‌کند. به صورت کلی می‌توان تاثیر پژوهش‌های مبهم‌نگاری بر برنامه‌های اندرویدی را از سه دیدگاه بررسی کرد. در حالت اول مبهم‌نگاری‌هایی نظیر تغییر نام و یا اجرای کلاس‌های رمزگذاری شده، 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 برنامه 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 برنامه‌های 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.

۳-۵ مقایسه‌ی روش‌ها

با توجه به فصلی که گذشت می‌توان گفت که اکثر پژوهش‌های موجود در زمینه‌ی تشخیص برنامه‌های بازبسته‌بندی شده، 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 روش‌های ایستا، سعی در افزایش کارایی روش‌های پیشین دارند. علت این موضوع آن است که تحلیل ایستا پاسخ‌گوی طیف وسیعی از برنامه‌های اندرویدی خصوصاً برنامه‌های رایگان است در حالی که پژوهش‌های موجود در زمینه‌ی تحلیل پویا، اکثراً از مجموعه‌ی داده‌ی آزمون کوچک و محدودی، خصوصاً در زمینه‌ی برنامه‌های تجاری، استفاده کرده‌اند. علت دیگر این موضوع آن است که تهدید اصلی کاربران در زمینه‌ی برنامه‌های اندرویدی بازبسته‌بندی شده، از سمت برنامه‌های رایگان رخ می‌دهد و برنامه‌های غیررایگان و تجاری، معمولاً 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 اما به صورت کلی، از آنجایی که الگوریتم‌های مقایسه‌ی گرافی، به خصوص الگوریتم‌های تشخیص گراف هم‌ریخت، سرعت بسیار پایینی در مقایسه دارند، معمولاً روش‌های این دسته به قدری کند هستند که عملاً استفاده از آن‌ها در یک 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. علاوه بر این، استفاده از روش‌هایی نظیر استفاده از دنباله‌ی دستورات و آپکدها، در کنار سرعت بالایی که دارند، دقت بسیار پایینی را ارائه می‌دهند به طوری که با کوچکترین تغییرات در برنامه، به طور کلی دنباله‌ی دستورات دچار تغییر خواهد شد.

| دسته‌بندی روش‌ها | پژوهش‌کنندگان | سربار محاسباتی | دقت تشخیص | مقایسه‌ی دودویی |
|----------------------------------|------------------|----------------|-----------|-----------------|
| مبتنی بر آپکد و دنباله‌ی دستورات | ژو و همکاران | | | |
| | ژو و همکاران | | | |
| | دزنور و همکاران | | | |
| | جرومه و همکاران | | | |
| | سرنیل و همکاران | | | |
| | لین و همکاران | | | |
| | فروکی و همکاران | | | |
| | کو و همکاران | | | |
| | کیشو و همکاران | | | |
| | راهول و همکاران | | | |
| مبتنی بر گراف | کروسل و همکاران | | | |
| | سان و همکاران | | | |
| | هو و همکاران | | | |
| | ژو و همکاران | | | |
| | چن و همکاران | | | |
| | آلدینی و همکاران | | | |
| | جنگ و همکاران | | | |
| | نگویان و همکاران | | | |
| | وانگ | | | |
| | ترکی | | | |
| مبتنی بر تحلیل ترافیک شبکه | | | | |
| مبتنی بر تحلیل منابع | | | | |

فصل ۴

نتایج جدید

در این فصل نتایج جدید به دست آمده در پایان نامه توضیح داده می شود. در صورت نیاز می توان نتایج جدید را در قالب چند فصل ارائه نمود. همچنین در صورت وجود پیاده سازی، بهتر است نتایج پیاده سازی را در فصل مستقلی پس از این فصل قرار داد.

فصل ۵

نتیجه‌گیری

در این فصل، ضمن جمع‌بندی نتایج جدید ارائه‌شده در پایان‌نامه یا رساله، مسائل باز باقی‌مانده و همچنین پیشنهادهایی برای ادامه‌ی کار ارائه می‌شوند.

فصل ۶

نتیجه‌گیری

- [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 the 30th Annual Computer Security Applications Conference, ACSAC '14*, page 56–65, New York, NY, USA, 2014. Association for Computing Machinery.
- [2] M. S. Bhatt, H. Patel, and S. Kariya. A Survey Permission Based Mobile Malware Detection. *Int.J.Computer Technology and Applications*, 6(5):852–856, 2015.
- [3] N. T. Cam, N. H. Khoa, T. T. An, N. P. Bach, and V.-H. Pham. Detect repackaged android applications by using representative graphs. In *2021 8th NAFOSTED Conference on Information and Computer Science (NICS)*, pages 102–106, 2021.
- [4] 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].
- [5] Play Protect | Google Developers — developers.google.com. <https://developers.google.com/android/play-protect>. [Accessed 02-Feb-2023].
- [6] 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].
- [7] 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].
- [8] 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. *Proceedings of the 2012*

7th Asia Joint Conference on Information Security, AsiaJCIS 2012, pages 62–69, 2012.

- [9] K. Khanmohammadi, N. Ebrahimi, A. Hamou-Lhadj, and R. Khoury. Empirical study of android repackaged applications. *Empirical Software Engineering*, 24(6):3587–3629, 2019.
- [10] T. Vidas and N. Christin. Sweetening android lemon markets: Measuring and combating malware in application marketplaces. *CODASPY 2013 - Proceedings of the 3rd ACM Conference on Data and Application Security and Privacy*, 2011:197–207, 2013.
- [11] 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.
- [12] Z. Ma, H. Wang, Y. Guo, and X. Chen. Libradar: Fast and accurate detection of third-party libraries in android apps. In *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*, pages 653–656, 2016.
- [13] 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.
- [14] V. Rastogi, Y. Chen, and X. Jiang. DroidChameleon: Evaluating Android anti-malware against transformation attacks. *ASIA CCS 2013 - Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*, pages 329–334, 2013.
- [15] Trail: The reflection api the javax; tutorials — docs.oracle.com. <https://docs.oracle.com/javase/tutorial/reflect/index.html>. [Accessed 02-Feb-2023].
- [16] X. Zhang, F. Breiting, 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.
- [17] ProGuard Manual: Home | Guardsquare — guardsquare.com. <https://www.guardsquare.com/manual/home>. [Accessed 02-Feb-2023].
- [18] Allatori Java Obfuscator — codedemons.net. <http://www.codedemons.net/allatori.html>. [Accessed 02-Feb-2023].

- [19] Y. Wang. Obfuscation-Resilient Code Detection Analyses for Android Apps. 2018.
- [20] 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.
- [21] L. Li, T. F. Bissyandé, J. Klein, and Y. Le Traon. An investigation into the use of common libraries in android apps. 1:403–414, 2016.
- [22] N. Karankar, P. Shukla, and N. Agrawal. Comparative study of various machine learning classifiers on medical data. pages 267–271, 2017.
- [23] Y. Shao, X. Luo, C. Qian, P. Zhu, and L. Zhang. Towards a scalable resource-driven approach for detecting repackaged android applications. *ACM International Conference Proceeding Series*, 2014-Decem(December):56–65, 2014.
- [24] F. Zhang, H. Huang, S. Zhu, D. Wu, and P. Liu. Viewdroid: Towards obfuscation-resilient mobile application repackaging detection. page 25–36, 2014.
- [25] 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.
- [26] 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.
- [27] 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.
- [28] A. Salem. Stimulation and Detection of Android Repackaged Malware with Active Learning. 2015.
- [29] T. Nguyen, J. T. McDonald, W. B. Glisson, and T. R. Aniel. Detecting repackaged android applications using perceptual hashing. *Proceedings of the Annual Hawaii International Conference on System Sciences*, 2020-January:6641–6650, 2020.
- [30] F. Alswaina and K. Elleithy. Android malware family classification and analysis: Current status and future directions. *Electronics (Switzerland)*, 9(6):1–20, 2020.

- [31] 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.
- [32] 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.
- [33] Q. Zhang, X. Zhang, Z. Yang, and Z. Qin. An efficient method of detecting repackaged android applications. pages 056 (4 .)–056 (4 .), 01 2014.
- [34] W. Zhou, Y. Zhou, X. Jiang, and P. Ning. Detecting repackaged smartphone applications in third-party android marketplaces. page 317–326, 2012.
- [35] A. Desnos. Android: Static analysis using similarity distance. pages 5394–5403, 2012.
- [36] 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.
- [37] B. B. Rad, M. Masrom, and S. Ibrahim. Opcodes histogram for classifying metamorphic portable executables malware. pages 209–213, 2012.
- [38] Q. Jerome, K. Allix, R. State, and T. Engel. Using opcode-sequences to detect malicious android applications. In *2014 IEEE International Conference on Communications (ICC)*, pages 914–919, 2014.
- [39] C. Liangboonprakong and O. Sornil. Classification of malware families based on n-grams sequential pattern features. pages 777–782, 2013.
- [40] 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.
- [41] 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 *2013 2nd International Conference on Advanced Computing, Networking and Security*, pages 247–252, 2013.

- [42] 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. page 336–341, 2013.
- [43] S. Kishore, R. Kumar, and S. Rajan. *Towards Accuracy in Similarity Analysis of Android Applications*, volume 11281 LNCS. Springer International Publishing, 2018.
- [44] 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.
- [45] J. Crussell, C. Gibler, and H. Chen. Scalable Semantics-Based Detection of Similar Android Applications. *Esorics*, pages 182–199, 2013.
- [46] W. Hu, J. Tao, X. Ma, W. Zhou, S. Zhao, and T. Han. Migdroid: Detecting app-repackaging android malware via method invocation graph. In *2014 23rd International Conference on Computer Communication and Networks (ICCCN)*, pages 1–7, 2014.
- [47] W. Zhou, Y. Zhou, M. Grace, X. Jiang, and S. Zou. Fast, scalable detection of ”piggybacked” mobile applications. *CODASPY 2013 - Proceedings of the 3rd ACM Conference on Data and Application Security and Privacy*, pages 185–195, 2013.
- [48] K. Chen, P. Liu, and Y. Zhang. Achieving accuracy and scalability simultaneously in detecting application clones on Android markets. *Proceedings - International Conference on Software Engineering*, (1):175–186, 2014.
- [49] 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.
- [50] J. Zheng, K. Gong, S. Wang, Y. Wang, and M. Lei. Repackaged apps detection based on similarity evaluation. pages 1–5, 2016.
- [51] M. Torki. Detecting repackaged android applications. 2018.
- [52] Y. Wang. Obfuscation-resilient code detection analyses for android apps,” phd thesis, the ohio state university. 2018.

- [53] 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.
- [54] 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.
- [55] 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.
- [56] A. Rodriguez and A. Laio. Clustering by fast search and find of density peaks. *Science*, 344(6191):1492–1496, 2014.
- [57] J. Malik and R. Kaushal. Credroid: Android malware detection by network traffic analysis. In *Proceedings of the 1st ACM Workshop on Privacy-Aware Mobile Computing*, PAMCO ’16, page 28–36, New York, NY, USA, 2016. Association for Computing Machinery.
- [58] D. Iland and A. Pucher. Detecting android malware on network level. 2011.
- [59] S. Kandukuru and R. M. Sharma. Android malicious application detection using permission vector and network traffic analysis. pages 1126–1132, 2017.
- [60] M. Lin, D. Zhang, X. Su, and T. Yu. Effective and scalable repackaged application detection based on user interface. In *2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 1–6, 2017.
- [61] 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 *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 323–333, 2018.
- [62] 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.
- [63] S. Li. Juxtapp and dstruct: Detection of similarity among android applications. 2012.
- [64] 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, *Web Information Systems and Applications*, pages 579–591, Cham, 2020. Springer International Publishing.

- [65] F. Lyu, Y. Lin, J. Yang, and J. Zhou. Suidroid: An efficient hardening-resilient approach to android app clone detection. In *2016 IEEE Trustcom/BigDataSE/ISPA*, pages 511–518, 2016.
- [66] X. Liu, Z. Yu, Z. Song, L. Chen, and Z. Qin. Mdsdroid: A multi-level detection system for android repackaged applications. In *2021 IEEE 6th International Conference on Signal and Image Processing (ICSIP)*, pages 1128–1133, 2021.
- [67] 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 *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*, pages 132–142, 2017.
- [68] 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.
- [69] V. Costamagna, C. Zheng, and H. Huang. Identifying and evading android sandbox through usage-profile based fingerprints. page 17–23, 2018.
- [70] Y. Zhang, K. Huang, Y. Liu, K. Chen, L. Huang, and Y. Lian. Timing-based clone detection on android markets. 12 2015.
- [71] T. Nguyen, J. T. McDonald, W. B. Glisson, and T. R. Anandel. Detecting repackaged android applications using perceptual hashing. 2020.
- [72] 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.
- [73] T. Bläsing, L. Batyuk, A.-D. Schmidt, S. A. Camtepe, and S. Albayrak. An android application sandbox system for suspicious software detection. In *2010 5th International Conference on Malicious and Unwanted Software*, pages 55–62, 2010.
- [74] D. Kim, A. Gokhale, V. Ganapathy, and A. Srivastava. Detecting plagiarized mobile apps using API birthmarks. *Automated Software Engineering*, 23(4):591–618, 2016.
- [75] Q. Guan, H. Huang, W. Luo, and S. Zhu. Semantics-based repackaging detection for mobile apps. In J. Caballero, E. Bodden, and E. Athanasopoulos, editors, *Engineering Secure Software and Systems*, pages 89–105, Cham, 2016. Springer International Publishing.

- [76] 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 the 2015 International Symposium on Software Testing and Analysis, ISSTA 2015*, page 71–82, New York, NY, USA, 2015. Association for Computing Machinery.
- [77] O. S. J. Nisha and S. M. S. Bhanu. Detection of repackaged android applications based on apps permissions. In *2018 4th International Conference on Recent Advances in Information Technology (RAIT)*, pages 1–8, 2018.
- [78] 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*, page 1–12, New York, NY, USA, 2013. Association for Computing Machinery.
- [79] Y. Zhang and K. Chen. Appmark: A picture-based watermark for android apps. In *2014 Eighth International Conference on Software Security and Reliability (SERE)*, pages 58–67, 2014.
- [80] 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*, page 635–646, New York, NY, USA, 2014. Association for Computing Machinery.
- [81] X. Sun, J. Han, H. Dai, and Q. Li. An active android application repacking detection approach. In *2018 10th International Conference on Communication Software and Networks (ICCSN)*, pages 493–496, 2018.
- [82] L. Luo, Y. Fu, D. Wu, S. Zhu, and P. Liu. Repackage-proofing android apps. In *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 550–561, 2016.
- [83] Q. Zeng, L. Luo, Z. Qian, X. Du, and Z. Li. Resilient decentralized android application repackaging detection using logic bombs. In *Proceedings of the 2018 International Symposium on Code Generation and Optimization, CGO 2018*, page 50–61, New York, NY, USA, 2018. Association for Computing Machinery.
- [84] 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, *Foundations and Practice of Security*, pages 189–204, Cham, 2020. Springer International Publishing.

- [85] 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*, page 222–235, New York, NY, USA, 2018. Association for Computing Machinery.
- [86] 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].
- [87] 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.
- [88] 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.
- [89] 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. Association for Computing Machinery.
- [90] 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.
- [91] 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, *Security and Privacy in Communication Networks*, pages 214–234, Cham, 2019. Springer International Publishing.
- [92] L. Zhang, H. Meng, and V. L. L. Thing. Progressive control flow obfuscation for android applications. In *TENCON 2018 - 2018 IEEE Region 10 Conference*, pages 1075–1079, 2018.
- [93] 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*, page 329–334, New York, NY, USA, 2013. Association for Computing Machinery.

- [94] 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.

واژه‌نامه

الف

ابتکاری heuristic.....
 ابعاد بالا high dimensions.....
 اریب bias.....
 آستانه threshold.....
 اصل لانه‌ی کبوتری pigeonhole principle.....
 ان‌پی-سخت NP-Hard.....
 انتقال transition.....

ت

تجربی experimental.....
 تراکم density.....
 تقریب approximation.....
 تقسیم‌بندی partition.....
 توری mesh.....
 توزیع‌شده distributed.....

ب

برخط online.....
 برنامه‌ریزی خطی linear programming.....
 بهینه optimum.....
 بیشینه maximum.....

ج

جدایپذیر separable.....
 جعبه سیاه black box.....
 جویبار داده data stream.....

پ

پرت outlier.....
 پرسمان query.....
 پوشش cover.....
 پیچیدگی complexity.....

ح

حدی extreme.....
 حریصانه greedy.....

خ

خوشه cluster.....
 خطی linear.....

د

داده data
داده‌کاوی data mining
داده‌ی پرت outlier data
دوبرابر سازی doubling
دودویی binary

ف

فاصله distance
فضا space

ق

قطعی deterministic

ر

رأس vertex
رسمی formal

ک

کارا efficient
کاندیدا candidate
کمینه minimum

ز

زیرخطی sublinear

م

مجموعه set
مجموعه هسته coreset
سطح planar
موازی سازی parallelization
میان گیر buffer

س

سرشکن amortized
سلسله مراتبی hierarchichal

ش

شبه کد pseudocode
شیء object

ن

نابه جایی inversion
ناوردا invariant
نقطه‌ی مرکزی center point
نیم فضا half space

ص

صدق پذیری satisfiability

ه

هزینه‌ی آشوب price of anarchy (POA)

غ

غلبه dominate

ی

یال edge

پیوست آ

مطالب تکمیلی

پیوست‌های خود را در صورت وجود می‌توانید در این قسمت قرار دهید.

Abstract

We present a standard template for typesetting theses in Persian. The template is based on the `XYLATEX Persian` package for the `LATEX` typesetting system. This write-up shows a sample usage of this template.

Keywords: Thesis, Typesetting, Template, `XYLATEX Persian`



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