



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 مبهم‌سازی	۲
۷	۲-۱-۱ روش‌های ساده	۲
۸	۲-۱-۲ روش‌های میانی	۲
۱۰	۲-۱-۳ روش‌های خاکستری	۲
۱۱	۲-۱-۴ روش‌های ترکیبی	۲
۱۱	۲-۱-۵ انواع مبهم‌نگارها	۲
۱۲	۲-۲ ساختار فایل‌های برنامه‌های اندرویدی	۲
۱۴	۲-۳ کتابخانه‌های اندرویدی	۲
۱۴	۲-۴ طبقه‌بندی	۲
۱۵	۲-۵ بازبسته‌بندی برنامه‌های اندرویدی	۲
۱۶	۳ تعریف مسئله و مرور کارهای پیشین	۳
۱۷	۳-۱ تعریف مسئله	۳
۱۸	۳-۲ روند کلی تشخیص برنامه‌های بازبسته‌بندی شده	۳
۱۸	۳-۲-۱ پیش‌پردازش برنامه‌های اندرویدی	۳
۱۹	۳-۲-۲ استخراج ویژگی	۳

۲۰	۳-۲-۳ تشخیص بازبسته‌بندی
۲۱	۳-۳ مرورکارهای پیشین
۲۱	۳-۳-۱ مبتنی بر تحلیل ایستا
۲۳	۴ نتایج جدید
۲۴	۵ نتیجه‌گیری
۲۵	۶ نتیجه‌گیری
۲۶	مراجع
۳۰	واژه‌نامه
۳۲	آ مطالب تکمیلی

فهرست جدول‌ها

فهرست شکل‌ها

- ۸ ۲-۱ نمونه‌ای از مبهم‌نگاری با استفاده از تغییر نام شناسه‌ها
- ۲-۲ نمونه‌ای از مبهم‌نگاری با استفاده از قابلیت بازتاب به منظور پنهان‌سازی واسطه فراخوانی شده
- ۱۰ batteryinfo به نام
- ۱۳ ۲-۳ ساختار پوشه‌ها و فایل‌های بسته‌های Apk [۱]

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 [۲] 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 [۳] 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 [۴]. 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 [۵]. مطابق پژوهش آقای ژو و همکاران [۶] 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 [۷]، 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 [۸] 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 [۹]. 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}
- Resourcues^{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 [۱۰] 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 شباهت‌سنجی، کدهای کتابخانه‌ای تشخیص داده می‌شوند. مشخص است که روش‌های مبتنی بر شباهت‌سنجی 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 شباهت‌سنجی، 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 شباهت‌سنجی 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[۱۱]:

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 معنانشناسی^۳ تغییر می‌کند و ویژگی‌های مبتنی بر نحو^۴ ثابت باقی می‌مانند. بنابراین، روش‌هایی که بیشتر مبتنی بر معنانشناسی برنامه‌های اندرویدی هستند، دچار خطای بیشتری در این سطح از مبهم‌نگاری می‌شوند. در ادامه به معرفی مختصری از انواع روش‌های مبهم‌نگاری مطابق با پژوهش [۱۲] در این دسته می‌پردازیم:

- **تغییر نام شناسه‌ها:** تغییر نام شناسه‌های موجود در برنامه شامل نام کلاس‌ها، متدها و یا متغیرها^۵ی موجود [۱۱]

```
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 معنانشناسی 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 نُپ^{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 [۱۳] 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 [۱۴] 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. [۱۵]

• 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. [۱۶، ۱۷]

• 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. [۱۸]

• **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

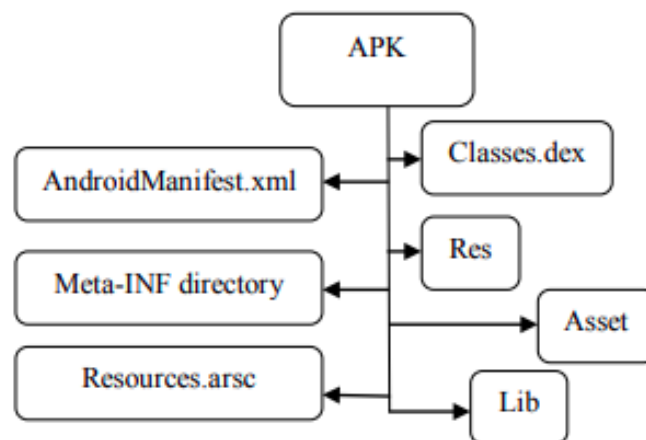
^{۳۰} Resource Id

^{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 [۱]

^{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 [۱۹]. 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 [۲۰]. 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.

^{۳۸}Algorithm

^{۳۹}Sorting

^{۴۰}Spam

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 [۲۱، ۲۲] 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 [۲۳، ۲۴] 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 [۲۵] بازبسته‌بندی را منوط به تغییر در امضای برنامه می‌دانند اما بسیاری از پژوهش‌های به‌روزتر، نظیر [۲۶، ۲۷] 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 برنامه‌های مورد نظر است 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 بازسته‌بندی 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 بازسته‌بندی شده است و تمرکز پژوهش بر روی تشخیص کدهای کتابخانه‌ای نبوده است، در ابتدا روند کلی تشخیص برنامه‌های بازسته‌بندی 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 بازسته‌بندی شده می‌پردازیم و مسئله‌ی تشخیص برنامه‌های بازسته‌بندی 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 بازسته‌بندی، هنوز تعریف مشخصی نیز برای تشخیص بازسته‌بندی ارائه نشده است. پژوهش‌های سال‌های اخیر در حالت کلی تشخیص بازسته‌بندی را به دو صورت تعریف می‌کنند:

تعریف ۳-۱ (تشخیص بازسته‌بندی مبتنی بر برنامه‌ی مبدا) / تشخیص بسته‌ی بازسته‌بندی 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 بازسته‌بندی شده است یا خیر و در صورتی که بود، جفت برنامه‌ی آن درون مخزن نیز مشخص می‌شود.

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

در سال‌های اخیر، اکثر پژوهش‌ها از یکی از تعاریف بالا برای تشخیص بازسته‌بندی استفاده کرده‌اند.

برای پاسخ به تعریف ۲، پژوهش‌هایی نظیر [۲۸، ۲۹، ۳۰] 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 [۳۱، 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 مرورکارهای پیشین

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

استفاده از آپکد^۹‌های موجود در فایل‌های دالویک، یکی از روش‌های تشخیص برنامه‌ک‌های بازبسته‌بندی شده است. هدف از پژوهش آقای ژو [۳۲] 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 غیررسمی 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 `Hash` 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 ۲۲۹۰۶ برنامه موجود در فروشگاه‌های رسمی استفاده شده و نتایج پژوهش نشان می‌دهد که ۵ تا ۱۳ درصد از برنامه‌های موجود در فروشگاه‌های غیررسمی، بازبسته‌بندی شده‌ی برنامه‌های فروشگاه‌های رسمی است. در پژوهش دیگری که توسط آقای ژو [۲۴] 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.

...ادامه

^{۱۱} MetaData

^{۱۲} Sliding Window

^{۱۳} Edit Distance

^{۱۴} Manifest

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] 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.
- [2] 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].
- [3] Play Protect | Google Developers — developers.google.com. <https://developers.google.com/android/play-protect>. [Accessed 02-Feb-2023].
- [4] 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].
- [5] 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].
- [6] 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.
- [7] K. Khanmohammadi, N. Ebrahimi, A. Hamou-Lhadj, and R. Khoury. Empirical study of android repackaged applications. *Empirical Software Engineering*, 24(6):3587–3629, 2019.
- [8] 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.
- [9] 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.
 - [10] 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.
 - [11] 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.
 - [12] 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.
 - [13] Trail: The reflection api the javax; tutorials — docs.oracle.com. <https://docs.oracle.com/javase/tutorial/reflect/index.html>. [Accessed 02-Feb-2023].
 - [14] X. Zhang, F. Breitingner, 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.
 - [15] ProGuard Manual: Home | Guardsquare — guardsquare.com. <https://www.guardsquare.com/manual/home>. [Accessed 02-Feb-2023].
 - [16] Allatori Java Obfuscator — codedemons.net. <http://www.codedemons.net/allatori.html>. [Accessed 02-Feb-2023].
 - [17] Y. Wang. Obfuscation-Resilient Code Detection Analyses for Android Apps. 2018.
 - [18] 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.
 - [19] 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.

- [20] N. Karankar, P. Shukla, and N. Agrawal. Comparative study of various machine learning classifiers on medical data. pages 267–271, 2017.
- [21] 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.
- [22] F. Zhang, H. Huang, S. Zhu, D. Wu, and P. Liu. Viewdroid: Towards obfuscation-resilient mobile application repackaging detection. page 25–36, 2014.
- [23] 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.
- [24] 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.
- [25] 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.
- [26] A. Salem. Stimulation and Detection of Android Repackaged Malware with Active Learning. 2015.
- [27] 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.
- [28] F. Alswaina and K. Elleithy. Android malware family classification and analysis: Current status and future directions. *Electronics (Switzerland)*, 9(6):1–20, 2020.
- [29] 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.
- [30] 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.

- [31] Q. Zhang, X. Zhang, Z. Yang, and Z. Qin. An efficient method of detecting repackaged android applications. pages 056 (4 .)–056 (4 .), 01 2014.
- [32] W. Zhou, Y. Zhou, X. Jiang, and P. Ning. Detecting repackaged smartphone applications in third-party android marketplaces. page 317–326, 2012.

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
 ناورد 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