دوفصل‌نامه علمی منادی امنیت فضای تولید و تبادل اطلاعات (افتا)
سال ۱۴۰۰، جلد ۱۹، شماره ۲ (۴۲-۵۰)
https://monadi.isc.org.ir

ارائه‌شده در هفدهمین کنفرانس بین المللی انجمن رمز ایران در دانشگاه علم و صنعت ایران، ۱۹ و ۲۰ شهریور ۱۳۹۹

# تشخیص نشت اطلاعات در برنامه‌های آندرویدی با کمک وارسی مدل★

فاطمه عسکری نیسیانی* و بهروز ترک لادانی

دانشکده مهندسی کامپیوتر، دانشگاه اصفهان، اصفهان، ایران

**اطلاعات مقاله**

کلمات کلیدی:
برنامه‌های آندرویدی
نشت حریم خصوصی
تحلیل ردیابی اجرا
وارسی مدل

dor: 10.0000/000000000

نوع مقاله: پژوهشی

**چکیده**

سیستم‌عامل آندروید به دلیل وجود توسعه‌دهندگان فعال و فراگیر بودن، کاربران زیادی را به خود جذب کرده است. این در حالی است که بسیاری از برنامه‌های نوشته شده برای این محیط، دقت کافی در حفظ حریم خصوصی کاربران به خرج نمی‌دهند و به آسانی منجر به نشت اطلاعات حساس کاربران می‌شوند. بنابراین، یک چالش اصلی این است که چطور می‌توان به‌صورت مؤثری چنین آسیب‌پذیری‌هایی را شناسایی کرد. در این مقاله رویکردی مبتنی بر چارچوب وارسی مدل به منظور تشخیص امکان نشت اطلاعات حساس در برنامه‌های آندرویدی از طریق وارسی رفتار برنامه‌ها ارائه می‌کنیم. برای این کار ابزار JPF-Android را برای سازگار کردن با مسئله موردنظر توسعه داده و از آن برای مدل‌سازی و وارسی نشت اطلاعات در برنامه‌های آندرویدی استفاده می‌کنیم. برای ارزیابی رویکرد ارائه‌شده، توانایی آن در تشخیص نشت اطلاعات را به صورت عملی با ارزیابی و با رویکردهای مشابه مقایسه کرده‌ایم. نتایج آزمایش‌های انجام شده حاکی از آن است که رویکرد پیشنهادی به طور موفقیت‌آمیزی قادر به تشخیص نشت اطلاعات است. به علاوه ابزار تهیه‌شده گزارش مفصلی را به توسعه‌دهندگان ارائه می‌دهد تا اطلاعات دقیقی از نحوه انجام نشت اطلاعات به دست آورند.

© ۱۴۰۰ انجمن رمز ایران

## ۱ مقدمه

ویژگی‌های جذاب گوشی‌های هوشمند (مثل تحرک‌پذیری، امکان اتصال به اینترنت، کاربرپسند بودن، سهولت انجام عملیات، امکان اشتراک‌گذاری اطلاعات و انجام معاملات مالی) افراد زیادی (از یک طرف توسعه‌دهندگان نرم‌افزار و از طرف دیگر مهاجمان) را به خود جذب کرده است که هدف اصلی آن‌ها کسب درآمد است.

تحقیقات اخیر نشان داده است که درصد زیادی از برنامه‌های گوگل پلی (حدود ۷۰٪ برنامه‌ها تا سال ۲۰۱۷) نمی‌توانند الزامات کیفی را برآورده کنند [۱]؛ بنابراین درصورتی‌که برخی اشکالات[1] کدنویسی یا طراحی توسط مهاجمان کشف شود، این اشکالات می‌تواند منجر به آسیب‌پذیری قابل بهره‌برداری شود. آسیب‌پذیری از نظر مفهومی متفاوت از اشکالات سنتی است، چرا که آسیب‌پذیری نمایانگر سوءاستفاده از عملکرد امنیتی است درحالی‌که اشکالات سنتی، عملکردهای اشتباه غیرامنیتی است [۲]. به‌عنوان‌مثال، بر اساس آمار، در سال ۲۰۱۸، مهاجمین قادر بوده‌اند از ۸۹٪ آسیب‌پذیری‌ها حتی بدون دسترسی مستقیم سوءاستفاده کنند [۳]. وجود برخی از این آسیب‌پذیری‌ها حریم خصوصی کاربران نهایی را مورد هدف قرار می‌دهند و منجر به نشت اطلاعات حساس و نقض حریم خصوصی کاربران می‌شوند.

تحلیل برنامه‌های آندرویدی یکی از روش‌هایی است که توسط توسعه‌دهندگان بازارهای شخص ثالث و کاربران نهایی برای تشخیص

[1] bugs

★از کمیته علمی هفدهمین کنفرانس بین‌المللی انجمن رمز ایران برای داوری این مقاله تشکر می‌شود.
*نویسنده مسئول
آدرس‌های رایانامه: askari@eng.ui.ac.ir (فاطمه عسکری نیسیانی)، ladani@eng.ui.ac.ir (بهروز ترک لادانی)
© ۱۴۰۰ تمامی حقوق متعلق به انجمن رمز ایران است.

چنین آسیب‌پذیری‌هایی مورد استفاده قرار می‌گیرد. این کار می‌تواند امنیت برنامه‌های آندرویدی را افزایش دهد و سوءاستفاده از اطلاعات کاربران را کاهش دهد.

رویکرد پیشنهادی در این مقاله، یک چارچوب تحلیل ترکیبی است که نشت حریم خصوصی برنامه‌های آندرویدی را با استفاده از بررسی‌کننده مدل JPF انجام می‌دهد. وارسی مدل یک تکنیک وارسی خودکار است که تمام حالات ممکن سیستم را به شیوه‌ی سامانمند بررسی می‌کند تا ببیند آیا ویژگی موردنظر برآورده می‌شود یا خیر. فرآیند وارسی مدل شامل مدل‌سازی یک سیستم تحت بررسی، مشخص کردن ویژگی و در نهایت وارسی ویژگی‌های موردنظر روی مدل است. مزایای استفاده از بررسی‌کننده JPF عبارت‌اند از پشتیبانی برنامه‌های هم‌روند، نخ‌های جاوا و متدهای انعکاسی [1] که در اکثر رویکردهای قبلی پشتیبانی نمی‌شود. علاوه بر آن JPF برنامه تحت بررسی را اجرا می‌کند و با جستجوی کامل فضای حالت قادر به اثبات عدم وجود نقض ویژگی است؛ بنابراین نیازی به تجهیز [2] کردن برنامه ندارد، چرا که تجهیز کد اغلب وقت‌گیر و مستعد خطا است [۴].

البته JPF به منظور وارسی برنامه‌های آندرویدی با چالش مدل‌سازی محیطی آندروید روبرو است. مدل‌سازی محیطی برنامه‌های آندروید، راهی برای کنترل رفتار چنین برنامه‌هایی فراهم می‌کند. مدل‌سازی این محیط‌ها پیچیده است، چرا که از یک طرف کتابخانه‌های زیادی دارد و از طرف دیگر رویداد محور است. پیاده‌سازی سامانه‌های نرم‌افزاری برای تحلیل این گونه محیط‌ها نیز دشوار است، چرا که از یک طرف باید بر پیچیدگی مدل‌سازی غلبه کند و از طرف دیگر باید بتواند با دقت بالا تمامی حالات را بررسی کند. برای غلبه بر این چالش‌ها، در رویکرد پیشنهادی از وارسی مدل استفاده می‌شود.

به منظور حل این چالش، چندین کار تحقیقاتی به مدل‌سازی چارچوب آندروید برای بررسی‌کننده JPF پرداخته‌اند. ما با بررسی چارچوب‌های ارائه‌شده در این زمینه، به علت مزایای موجود در JPF-Android [۵] یا به اختصار JPA، از آن (با انجام برخی تغییرات و توسعه‌ها) برای مدل‌سازی چارچوب آندروید استفاده می‌کنیم. JPA یک چارچوب مبتنی بر JPF است که در سال ۲۰۱۷ توسط هیلا و همکارانش ارائه شد [۶]. JPA، چارچوب آندروید (برای پردازش ورودی و مدیریت چرخه حیات مؤلفه‌ها)، خدمات سیستم آندروید (مدیریت منابع، مدیریت پنجره، مدیریت فعالیت چندرسانه‌ای، پایگاه داده، مدیریت ورودی/خروجی، درخواست HTTP، رابط گرافیکی کاربر)، مجموعه‌ای از متدهای پس فراخوان، مدیریت رویدادهای کاربری/سیستمی و همچنین مسیرهایی از اینتنت‌های دریافت شده توسط مؤلفه را پیاده‌سازی می‌کند [۶]. علاوه بر آن به توسعه‌دهندگان کمک می‌کند تا برنامه‌های آندرویدی را اجرا کنند و خطاهای منطقی (مثل بن‌بست، استثناهای زمان اجرا و نقض ویژگی‌های تعریف‌شده توسط کاربر) که منجر به رفتار ناخواسته و غیره منتظره می‌شود را شناسایی کنند.

[1] reflection method [2] instrument

با توجه به مسائل مربوط به مقیاس‌پذیری، لازم است وارسی مدل با تکنیک‌های کاهش سفارش جزئی (مثل قطعه‌قطعه [3] کردن، جستجوی ابتکاری [4]، وارسی مدل پیمانه‌ای و تکنیک‌های تحلیل ایستا و پویا) ترکیب شود [۷]. ما نیز در رویکرد پیشنهادی برای استخراج مدل محیطی از تکنیک تحلیل ایستا استفاده می‌کنیم. استخراج مدل محیطی برای هر برنامه آندرویدی یک چالش مهم است، چرا که باید به اندازه‌ی کافی کلی باشد تا اجازه‌ی تحلیل بخش‌های بزرگی از رفتارهای ممکن یک سیستم را بدهد. به‌منظور خودکارسازی فرآیند استخراج مدل از OCSEGEN [۷] به همراه گسترشی که در آن انجام داده‌ایم استفاده‌شده است. چارچوب OCSEGEN، مبتنی بر Soot (یک ابزار بهینه‌سازی کد جاوا) [۸] است که از تحلیل جریان داده درون رویه‌ای استفاده می‌کند تا اطلاعات لازم برای هر محیط را محاسبه کند. به عبارت دیگر با تحلیل بایت‌کد جاوا برای مؤلفه‌ها و سیستم‌های تحت بررسی (به ویژه برنامه‌هایی با رابط‌های گرافیکی کاربر و برنامه‌های وب) که با جاوا نوشته‌شده است، مدل محیطی تولید می‌کند. اگر چه اغلب برنامه‌های آندرویدی با جاوا نوشته شده است اما آن‌ها دو تفاوت عمده با برنامه‌های جاوا دارند که می‌توان به رویداد محوری و داشتن نقاط ورودی زیاد اشاره کرد. بر همین اساس در رویکرد پیشنهادی تغییراتی روی OCSEGEN اعمال شده است.

به‌طور خلاصه در این مقاله با استفاده از گسترش چارچوب OCSEGEN، JPA و افزودن امکان ردیابی جریان به موتور اکتشاف JPF به‌صورت خودکار یک برنامه آندرویدی را به منظور تشخیص نشت اطلاعات روی JPF اجرا کند و نتایج حاصل از بررسی را گزارش دهد.

ساختار این مقاله بدین شکل است. در بخش ۲ مروری بر کارهای مرتبط خواهیم داشت. سپس در بخش ۳ مسئله تحقیق و در بخش ۴ روش پیشنهادی شامل مدل‌سازی محیطی (انجام تغییرات در OCSEGEN و JPA) و تشریح ویژگی افزوده‌شده به موتور JPF ارائه‌شده است. در نهایت در بخش ۵، رویکرد پیشنهادی را ارزیابی نموده، با سایر کارهای ارائه شده در این زمینه مقایسه می‌کنیم. بخش ۶ جمع‌بندی مقاله را ارائه می‌کند.

## ۲ کارهای مرتبط

رویکردهای ارائه شده در تحلیل و تشخیص نشت اطلاعات برنامه‌های آندرویدی را می‌توان به سه دسته ایستا، پویا و ترکیبی تقسیم کرد. در ادامه ابزارهای مرتبط با هر دسته از تحلیل برنامه‌ها که برای تشخیص نشت اطلاعات به‌کاررفته است، توضیح داده می‌شود.

FlowDroid [۹] یک ابزار تحلیل ایستای پوششی است که نشت حریم خصوصی در برنامه‌های آندرویدی را با تحلیل جریان داده در هر مؤلفه آن انجام می‌دهد. مدل چارچوب از یک متد اصلی تصنعی [5] برای به دست آوردن توالی‌های متد چرخه حیات در آندروید استفاده می‌کند. علاوه بر آن، یک تحلیل «نام تصنعی» مبتنی بر گراف فراخوان با استفاده از Soot انجام می‌دهد. پس می‌توان گفت این ابزار از مدل محیطی سطح

[3] slicing [4] heuristic search [5] dummy main method

برنامه حمایت می‌کند. این ابزار حساس به جریان، متن و شیء است اما ارتباط بین مؤلفه‌ای را کنترل نمی‌کند و نمی‌تواند به مسائل امنیتی مربوط به آن بپردازد.

Dexteroid [۱۰]، یک ابزار تحلیل ایستا است که از مدل‌های چرخه حیات برای تشخیص نشت اطلاعات و رفتارهای بدخواهانه که هدف آن‌ها حریم خصوصی است، استفاده می‌کند. با دریافت فایل APK آندروید، بایت‌کدهای جاوا را به همراه فایل مانیفست استخراج می‌کند. سپس ضمن تحلیل هر یک از این مؤلفه‌ها، دنباله‌های پس فراخوان مربوطه را استخراج کرده و به همراه پس فراخوان‌های چرخه حیات آندروید مدلی از برنامه کاربردی ایجاد می‌کند. درواقع این رویکرد تنها بر پس فراخوان‌های چرخه حیات تمرکز دارد و انواع دیگر از فراخوانی‌های ضمنی ناشی از فهرست چارچوب آندروید را کنترل نمی‌کند؛ بنابراین این رویکرد از دقت کافی برخوردار نیست، چرا که بدافزارها می‌توانند به آسانی پشت فراخوانی‌های ضمنی پنهان شوند.

TaintDroid [۱۱]، یکی از اولین و پیچیده‌ترین سامانه‌های ردیابی جریان داده پویا است که از ماشین مجازی دلویک استفاده می‌کند و به کاربران اجازه می‌دهد تا جریان‌های داده حساس را ردیابی و تحلیل کنند و به طور بالقوه برنامه‌هایی که حاوی نشت حریم خصوصی است را تشخیص دهند. این چارچوب به عبارتی عملکرد سیستم‌عامل آندروید را گسترش می‌دهد تا جریان‌های داده محرمانه و آسیب‌پذیر را از طریق برنامه نصب‌شده ثبت کند. این رویکرد قادر به ردیابی کد بومی یا تمایز بین منابع اطلاعات حساس مختلف نیست.

DroidPF [۱۲] چارچوبی است که با استفاده از بررسی‌کننده مدل به صورت پویا فضای حالت واقعی برنامه آندرویدی را بررسی می‌کند تا آسیب‌پذیری‌های نشت حریم خصوصی و نشت مجوزها را در برنامه‌های آندرویدی شناسایی کند. درواقع این چارچوب، مدلی از سیستم عامل که شامل مؤلفه‌هایی برای تعامل کاربر و ورودی‌های محیطی است را پیاده‌سازی می‌کند تا اجرای پویای برنامه را هدایت کند. همچنین از یک راه‌انداز به‌منظور کاوش فضای حالت استفاده می‌کند. این چارچوب، قادر به ردیابی کد بومی و اعتبارسنجی ورودی نیست. علاوه بر آن جریان‌های ضمنی بین اطلاعات حساس را بررسی نمی‌کند؛ بنابراین این رویکرد از دقت کافی برخوردار نیست.

IIFDroid [۱۳] یک ابزار مبتنی بر Soot است که از تحلیل پوششی ایستا استفاده می‌کند تا نشت حریم خصوصی بین مؤلفه‌ای برنامه‌های آندروید را تشخیص دهد. این ابزار در بالای ICCTA قرارداد و گراف جریان کنترل بین مؤلفه‌ای را به عنوان ورودی دریافت می‌کند. به منظور استخراج جریان‌ها، از گراف وابستگی سیستمی استفاده می‌کند. سپس گراف را با متدهای مبدأ و چاهک حاشیه‌نویسی می‌کند. در نهایت گراف وابستگی سیستمی را در یک شیوه حساس به متن پیمایش رو به عقب انجام می‌دهد تا نشت اطلاعات بالقوه را تشخیص دهد. این رویکرد کنترل همروندی و انعکاس را پشتیبانی نمی‌کند.

AndroidLeaker [۱۴] برای تشخیص نشت اطلاعات برنامه‌های آندروید از رویکرد ترکیبی استفاده می‌کنند. به عبارت دیگر از تحلیل پوششی ایستا به منظور شناسایی نشت اطلاعات درون برنامه‌ای و از تحلیل پویا به منظور جلوگیری از هر گونه نشت ناشی از همکاری بین برنامه‌ها استفاده می‌کند. تحلیل ایستای این ابزار، مبتنی بر چارچوب Soot است و شامل تحلیل جریان داده و گراف فراخوان است و در تحلیل پویا، همه‌ی برنامه‌های تحت بررسی تجهیز می‌شوند. این ابزار به علت عدم مدل‌سازی چرخه حیات آندروید از دقت پایین‌تری نسبت به سایر روش‌ها برخوردار است.

ایده ما در این مقاله استفاده از رویکرد وارسی مدل و به طور خاص استفاده از ابزار وارسی مدل JPF است. JPF یک بررسی‌کننده مدل بسیار قوی و معروف برای کدهای جاوا است که در آن وارسی برنامه‌های هم‌روند، نخ‌های جاوا و متدهای انعکاسی پشتیبانی می‌شود. این موارد در اکثر رویکردهای قبلی پشتیبانی نمی‌شود. در ضمن با وجود اینکه هر چند بسیاری از برنامه‌ها از روش تجهیز کد برای تحلیل آسیب‌پذیری نشت اطلاعات استفاده کرده‌اند، ولی ساز و کار تجهیز اغلب بسیار وقت‌گیر و مستعد خطا است [۴]. در روش پیشنهادی چون JPF برنامه تحت بررسی را اجرا می‌کند و با جستجوی کامل فضای حالت قادر به اثبات عدم وجود نقض ویژگی است، بنابراین نیازی به تجهیز کد برنامه ندارد.

## ۳ تشریح مسئله

در این قسمت ابتدا به شرح و بیان مسئله می‌پردازیم. ویژگی که در این پژوهش موردبررسی قرار می‌گیرد، ویژگی تشخیص نشت اطلاعات است. تحلیل نشت اطلاعات اساساً یک مسئله امنیت جریان اطلاعات است. به عبارت دیگر، یک راه حل مطلوب برای شکست نشت اطلاعات، تشخیص منابعِ[1] حساسِ از پیش تعریف‌شده، دنبال کردن جریان اطلاعات از آن منابع و مسدود کردن آن‌ها در متدهای مقصد[2] است [۱۵]. به فراخوانی متدهای منبع که حداقل یک مقدار داده غیرثابت را به کد برنامه کاربردی برمی‌گرداند، متد منبع گویند و به فراخوانی متدهایی که حداقل یک مقدار داده غیرثابت از کد برنامه کاربردی را به‌عنوان ورودی بپذیرد، متدهای چاهک گویند. برای مثال متد getDeviceId()، متد منبع است چرا که مقدار متفاوت و غیرثابت IMEI را به برنامه برمی‌گرداند. علاوه بر آن، متد SendTextMessage، متد چاهک است چرا که متن پیام و شماره تلفن که به‌عنوان ورودی دریافت می‌کند، احتمالاً غیرثابت است [۱۶].

به منظور سهولت درک تشخیص نشت اطلاعات و انگیزه دادن به تحقیق، برنامه‌ی ساده Button4 از DroidBench [۱۷] را که در شکل ۱ نشان داده شده است را به عنوان مثال انگیزشی در نظر بگیرید. هنگامی که برنامه اجرا می‌شود، Button4 با یک فعالیت به صورت خودکار راه‌اندازی می‌شود و ابتدا حالت onCreate() چرخه حیات را فعال می‌کند. در این حالت شناسه منحصربه‌فرد دستگاه را می‌گیرد و درون فیلد ایستای imei

[1] sources [2] sinks

```
<Button xmlns:android="http://schemas.android.com/apk/res/android"
 android:text="@string/button"  android:onClick="sendMessage">
```

Button4

```
1.  public class Button4 extends Activity {
2.      private static String imei = null;
3.      @Override
4.      protected void onCreate(Bundle savedInstanceState) {
5.          super.onCreate(savedInstanceState);
6.          setContentView(R.layout.activity_button1);
7.          TelephonyManager telephonyManager = (TelephonyManager)
    getSystemService(Context.TELEPHONY_SERVICE);
8.          imei = telephonyManager.getDeviceId(); //source
9.      }
10.     public void sendMessage(View view){ ②
11.         SmsManager sms = SmsManager.getDefault();
12.         sms.sendTextMessage("+491254", null, imei, null, null);//sink
13.     }  ①
14.}
```

**شکل ۱.** مثال انگیزشی-برنامه‌ی Button4 از DroidBench

قرار می‌دهد. این فعالیت حاوی دکمه‌ای است با نام sendMessage است که به‌صورت ایستا در فایل طرح تعریف شده است. هنگامی که کاربر روی دکمه Button4 کلیک می‌کند، یک پیام کوتاه به شماره "491254" ارسال می‌کند و مقدار imei را درون آن قرار می‌دهد. وجود ویژگی نشت اطلاعات در خط ۱۳ رخ می‌دهد.

## ۴ راه حل پیشنهادی

همان‌طور که در بخش ۱ بیان شد، هدف این پژوهش، وارسی برنامه‌های آندرویدی به‌منظور تشخیص نشت اطلاعات با استفاده از بررسی‌کننده مدل JPF است. بررسی‌کننده مدل JPF مستقیماً با بایت‌کد جاوا کار می‌کند. بر همین اساس در رویکرد پیشنهادی از ابزار مهندسی معکوس اندروگارد استفاده شده است تا فایل اصلی برنامه را به بایت‌کد جاوا تبدیل کند. علاوه بر آن فایل‌های پیکربندی، طرح، منابع و سایر فایل‌ها را استخراج کرده و به همراه بایت‌کدهای جاوا در قالب پروژه‌ی آندرویدی با ساختار ANT یا Gradle در خروجی ارائه می‌دهد.

همان‌طور که گفته شد، وارسی مدل از دو بخش مدل‌سازی و ویژگی تشکیل‌شده است. بخش مدل‌سازی متکی بر دو نوع مدل (چارچوب آندروید و استخراج‌کننده مدل محیطی) است و بخش ویژگی نیز بر اساس دو روش (غیرحساس به جریان و حساس به جریان) پیاده‌سازی شده است که در ادامه به آن‌ها پرداخته می‌شود.

### ۱.۴ مدل‌سازی محیطی

گام اول فرآیند وارسی مدل، مدل‌سازی است چرا که هر سیستم تحت بررسی دارای وابستگی‌هایی است که برای اجرا روی بررسی‌کننده مدل به آن وابستگی‌ها نیاز دارد. به عبارت دیگر هر برنامه برای اجرا روی بررسی‌کننده مدل نیاز به مدل محیطی بسته و انتزاعی دارد. هدف از مدل محیطی، ایجاد محیطی است که برنامه در آن محیط قابل اجرا باشد.

در برخی از بررسی‌کننده‌های مدل، مشخصات سیستم با استفاده از زبان مدل‌سازی ایجاد می‌شوند و در برخی از بررسی‌کننده‌های دیگر (مثل JPF) از کد منبع به عنوان مدلی از سیستم استفاده می‌کنند [۱۸].

هر برنامه‌ی آندرویدی از چهار مؤلفه اصلی (فعالیت، سرویس، ارائه‌کننده محتوا و دریافت اطلاعیه) تشکیل شده است که هر یک چرخه حیات مربوط به خودش را دارد. علاوه بر این، برخلاف برنامه‌های جاوا که تنها از یک متد اصلی تشکیل شده‌اند، برنامه‌های آندرویدی حاوی چندین نقطه ورودی است؛ بنابراین باید توسط چارچوب آندروید فراخوانی و کنترل شوند. آندروید نیز هر مؤلفه برنامه را بر اساس چرخه حیات مربوطه مدیریت و کنترل می‌کند [۶]. همچنین برنامه‌های آندرویدی شامل مجموعه بزرگی از کتابخانه‌ها است. این کتابخانه‌ها حاوی فراخوانی‌هایی به کدهای بومی است که کدهای جاوای آن با استفاده از مدل کلاس‌ها و کدهای بومی مربوطه با کمک همتاهای بومی JPF پیاده‌سازی شده است. همتاهای بومی نقش کلیدی در واسط مدل جاوا دارند که این کلاس‌ها توسط ماشین مجازی جاوا در بالای ماشین مجازی جاوای میزبان اجرا می‌شوند. مدل کلاس‌ها، رفتار کلاس واقعی برنامه تحت بررسی را مدل می‌کند. به عبارت دیگر با اجرای کلاس‌های مدل، JPF مجبور می‌شود از نسخه‌های جایگزین کلاس‌های خاص جاوا استفاده کند و آن‌ها را بررسی مدل کند [۱۹].

برای کنترل رفتار برنامه‌های آندرویدی در بررسی‌کننده مدل نیاز به مدل‌سازی دقیق چارچوب آندروید است. بر همین اساس از JPA استفاده می‌شود؛ اما از آنجایی که برنامه‌های آندرویدی دارای وابستگی‌های زیادی است، JPA تمام وابستگی‌های آندرویدی را پیاده‌سازی نکرده است چرا که پیاده‌سازی همه وابستگی‌ها منجر به انفجار فضای حالت می‌شود. در رویکرد پیشنهادی برای افزایش مقیاس‌پذیری و کاهش انفجار فضای حالت، از تکنیک مبتنی بر Soot استفاده شده است تا تمام وابستگی‌های آندرویدی را خلاصه و استخراج کند. در واقع برای استخراج مدل محیطی، چارچوب OCSEGEN را برای برنامه‌های آندرویدی توسعه داده‌ایم. ازآنجاکه این موضوع بیشتر مربوط به نحوه پیاده‌سازی است و با توجه به محدودیت فضای مقاله از ورود بیشتر به جزئیات تغییرات اعمال شده اجتناب می‌کنیم.

### ۲.۴ مشخصات ویژگی

گام دوم فرآیند وارسی مدل، تعریف ویژگی برای تشخیص نشت اطلاعات است. این گام از دو روش غیر حساس به جریان و حساس به جریان تشکیل شده است. در روش غیر حساس به جریان، سعی شده از چارچوب JPA به‌منظور ردیابی جریان اطلاعات حساس استفاده شود و در روش حساس به جریان، ویژگی‌ای را به موتور جستجوی JPF می‌افزاید تا با استفاده از ردیابی جریان اطلاعات قادر به تشخیص نشت اطلاعات باشد.

### ۱.۲.۴ روش غیر حساس به جریان

JPA، سه فایل (چک‌لیست، اسکریپت و فایل ویژگی) را به‌عنوان ورودی دریافت می‌کند. چک‌لیست‌ها نوعی از مشخصه ویژگی هستند که در زمان اجرا، امکان وارسی برنامه‌ها را با JPF ساده‌تر می‌کند. چک‌لیست‌ها تنها حاوی توالی منظم از چک‌پوینت‌ها است که از چک‌پوینت‌ها می‌توان برای ردیابی اجرا در برنامه‌ها استفاده کرد تا ترتیبی که در آن متد فراخوانی شده است را وارسی کند. ایده ما این است که اولاً مسئله نشت اطلاعات یک مسئله جریان اطلاعات است، به عبارت دیگر در این روش، اطلاعات حساس را با شروع از یک منبع از پیش تعریف شده دنبال می‌کند تا به یک چاهک برسد و ثانیاً از چک‌پوینت‌ها می‌توان برای ردیابی اجرا در برنامه‌ها استفاده کرد. بنابراین، مسئله نشت اطلاعات را می‌توان با چک‌لیست که توالی منظم از چک‌پوینت‌ها است حل کرد.

از آنجایی که هر برنامه آندروید حاوی یک توالی از رویدادها است، JPA به منظور محدود کردن محیط و رفتار برنامه کاربردی، اسکریپت‌ها را ارائه کرده است. اسکریپت حاوی رویدادهای ورودی نامعینی است که می‌تواند به‌صورت نامعین بین پنجره‌ها سویچ کند. سومین فایل ورودی که باید به JPA داده شود، فایل ویژگی است. این فایل حاوی اطلاعاتی از قبیل فعالیت آغازگر برنامه، محل قرارگرفتن بایت‌کد کلاس‌ها، کد کلاس منبع جاوا، چک‌لیست‌ها، اسکریپت‌های ورودی، تنظیماتی به منظور پیکربندی JPF و تنظیماتی به منظور پیکربندی JPA است. در نهایت JPA بعد از دریافت فایل‌های ورودی، شروع به وارسی برنامه می‌کند. به عبارت دیگر در هر بار با فعال کردن یک رویداد، JPA، فایل ویژگی را در حین اجرای برنامه بررسی می‌کند تا ببیند آیا ویژگی روی برنامه صدق می‌کند یا خیر. بر همین اساس تشخیص نشت اطلاعات با استفاده از JPA دارای سه مرحله (ایجاد راه‌اندازی برای قرار دادن چک‌پوینت در برنامه‌ها، تعریف فایل چک‌لیست و فایل اسکریپت) است. ما در اولین گام، راه‌اندازی نوشته‌ایم که چک‌پوینت‌ها را در برنامه قرار دهد. در دومین گام، چک‌لیستی را ایجاد کرده و تمام حالات چیدمان چک‌پوینت در آن نوشته می‌شود. در سومین گام، فایل اسکریپتی به‌منظور مشخص کردن رویدادهای هر فعالیت ایجاد می‌کنیم.

### ۲.۲.۴ روش حساس به جریان

در این مرحله برای تشخیص نشت اطلاعات هنگام کاوش فضای حالت برنامه از ساز و کار ردیابی جریان اطلاعات درون شنونده استفاده می‌شود. شنونده، یکی از ساز و کارهایی است که JPF در اختیار کاربران قرار می‌دهد.

شنوندگان [1] [۱۹]، مهم‌ترین ساز و کار گسترش JPF است که به صورت پویا در زمان اجرا پیکربندی می‌شود. شنوندگان در همان سطح JPF اجرا می‌شوند و تغییری در هسته‌ی JPF اعمال نمی‌کنند؛ ولی راهی را برای مشاهده و تعامل با JPF فراهم می‌کنند. به عبارتی افزونه‌های زمان اجرا هستند که با JPF تعامل برقرار می‌کنند تا بر همه‌ی عملیات انجام‌شده توسط JPF نظارت کنند. شنوندگان با دو مؤلفه اصلی JPF (شنونده جستجو و ماشین مجازی) تعامل برقرار می‌کنند و اطلاعات مفیدی را در مورد گسترش JPF بازیابی می‌کنند. شنوندگان اصلی JPF، واسط‌های بزرگی هستند که اغلب، هر دو این واسط‌ها در اکثر پیاده‌سازی‌های شنونده استفاده می‌شود.

این مرحله به بررسی ویژگی نشت اطلاعات با استفاده از تحلیل ردیابی اجرا [2] می‌پردازد. از آنجایی که این روش به ردیابی جریان اطلاعات حساس درون برنامه‌ای می‌پردازد به آن حساس به جریان هم می‌گویند. درواقع با اجرای برنامه‌ها در سطح دستورالعمل‌های بایت‌کد، اطلاعات حساس (ممکن است توسط مؤلفه‌ها، واسط‌های برنامه‌نویسی و کلاس‌های مختلف مصرف شوند) را دنبال می‌کند و بررسی می‌کند که آیا سیاست نشت اطلاعات نقض می‌شود یا خیر [۴]. بر همین اساس تشخیص نشت اطلاعات از چهار مرحله (جمع‌آوری اطلاعات حساس، اجرای برنامه و اکتشاف فضای حالت، برچسب‌گذاری در حین اجرای برنامه و تحلیل نشت اطلاعات) تشکیل‌شده است. در اولین گام، واسط‌های برنامه‌نویسی حساس بر اساس مخزن SUSI [۱۶] مشخص می‌کند. سپس در گام دوم و سوم، هر یک از دستورالعمل‌های InstructionExecuted را اجرا می‌کند و هر مکانی که یکی از واسط‌های برنامه‌نویسی حساس فراخوانی می‌شود، برچسب منحصربه‌فردی را به آن محل اختصاص می‌دهد. در نهایت در گام چهارم، تمام حالات برنامه را بررسی می‌کند تا ببیند آیا اطلاعات حساس از مبدأ به چاهک نشت می‌کند یا خیر.

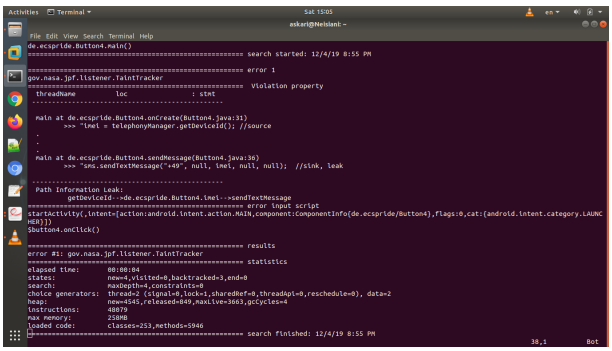
### ۳.۲.۴ اجرای برنامه و اکتشاف فضای حالت

در رویکرد پیشنهادی، JPF به‌عنوان موتور جستجو برای اجرا و اکتشاف فضای حالت برنامه در نظر گرفته شده است تا با استفاده از مؤلفه زمان‌بند انتخاب، تمام انتخاب‌های ممکن در هنگام اجرای برنامه را استخراج کند و بتواند فضای حالت برنامه را بررسی کند. البته به علت آن که برنامه‌های آندرویدی برخلاف جاوا از طریق فعال کردن رویدادها (شامل رویدادهای کاربری، سیستمی و چرخه حیات) کنترل می‌شوند، بنابراین نیاز به درایور یا کنترل‌کننده‌ای هست تا همه رویدادهای ممکن را شمارش کند و به JPF کمک کند تا فضای حالت برنامه‌های آندرویدی را کاوش کند [۶، ۱۲].

در رویکرد پیشنهادی از JPA به همراه گسترشی به‌منظور راه‌اندازی راه‌انداز مربوطه استفاده شده است. JPA، سه راهبرد مختلف (پیش‌فرض، ابتکاری و اسکریپت) برای تولید رویداد ارائه کرده است. از آنجایی که شمارش تمام جایگشت‌های رویداد ممکن است در بسیاری از توالی‌ها، رویدادهای غیرممکن را نیز تولید کند. برای جلوگیری از تولید رویداد در هر یک از راهبردها تمهیداتی در نظر گرفته شده است. برای مثال در روش پیش‌فرض، با استفاده از انطباق حالت، طول توالی‌های کشف‌شده را محدود می‌کند یا در روش ابتکاری، علاوه بر طول، تعداد دفعاتی که یک رویداد در هر مسیر رخ می‌دهد را محدود می‌کند. همچنین درروش

[1] listeners

[2] taint analysis




**شکل ۲.** خروجی رویکرد پیشنهادی از مثال انگیزشی

اسکریپت از کاربر خواسته شده است که محیط و رفتار برنامه کاربردی را در قالب اسکریپتی محدود کند [۶].

برای جلوگیری از انفجار فضای حالت چندین ساز و کار (پیش بارگذاری کلاس‌ها به همراه مقادیر پیش‌فرض، استفاده از حاشیه‌نویسی‌های JPF، محدود کردن طول توالی رویداد، پویایی شنونده در انتخاب‌های غیرقطعی، بهبود تکنیک‌های انتخاب غیرقطعی، تطابق حالت و روبه عقب JPF) در نظر گرفته‌شده است.

## ۵ ارزیابی و مقایسه با سایر روش‌ها

یک نمونه از خروجی حاصل از اعمال ابزار تهیه‌شده روی مثال انگیزشی مطرح‌شده در بخش ۳ در شکل ۲، نشان داده شده است. همان‌طور که مشاهده می‌شود، خروجی حاصل شامل نام نخ، پکیج، کلاسی که حاوی متد منبع یا چاهک است، شماره خط و دستورهای منبع و چاهک است. علاوه بر آن در قسمت پایین نیز آمارهایی از فضای حالت ارائه می‌دهد. این اطلاعات به تحلیل‌گر کمک می‌کند که دقیقاً متوجه دلایل بروز نشت اطلاعات و موقعیت دقیق آن در کد شود. برای ارزیابی دقت رویکرد پیشنهادی در مقایسه با سایر رویکردهای دیگر، از مجموعه برنامه‌های معیارهای DroidBench2 و ICCBench استفاده کردیم. این مجموعه‌ها ابزارها را در برابر چالش‌های مختلفی مثل حساسیت به شیء، فیلد، جریان، پس فراخوان‌های چرخه حیات، حلقه‌ها، ذخیره‌سازی در آرایه‌ها، لیست‌ها و درهم‌سازی‌ها ارزیابی می‌کند. ما به جز روش پیشنهادی، سه ابزار دیگر DroidSafe(DS)، DroidPF(DPF) و FlowDroid(FD) را نیز راه‌اندازی نموده و بر روی برنامه‌های محک معیار اعمال نمودیم. به عنوان نمونه جدول ۱ بخشی از ارزیابی انجام‌شده روی مجموعه برنامه‌های Inter-Component Communication از این محک معیار را نشان می‌دهد.

نتایج نهایی حاصل از مقایسه ابزار تهیه‌شده با ابزارهای مشابه بر اساس محک معیارهای DroidBench2 و ICCBench در جدول ۲ و ۳ نشان داده شده است. در این جدول معیارهای ارزیابی دقت شامل Precision، Recall، و F-Measure نشان داده شده است. همان‌گونه که مشاهده می‌شود، رویکرد ارائه شده از دقت بالاتری نسبت به روش‌های مشابه برخوردار است. دلیل این امر عمدتاً به عدم حمایت ابزارهای مورد مقایسه از ویژگی‌های خاص برنامه‌های هم‌روند، نخ‌های جاوا و متدهای

**جدول ۱.** نتیجه اعمال ابزارها روی بخشی از DroidBench2

| نام برنامه | FD | DPF | DS | رویکرد ما |
|---|---|---|---|---|
| ActivityCommunication1 | ✓ | ✓ | ✓ | ✓ |
| ActivityCommunication2 | ✓ | ✓ | ✓ | ✓ |
| ActivityCommunication3 | ✓ | ✓ | ✓ | ✓ |
| ActivityCommunication4 | ✓ | ✓ | ✓ | ✓ |
| ActivityCommunication5 | ✓ | ✓ | ✓ | ✓ |
| ActivityCommunication6 | ✓ | ✓ | ✓ | ✓ |
| ActivityCommunication7 | ✓ | ✓ | ✓ | ✓ |
| ActivityCommunication8 | ✓ | ✓ | ✓ | ✓ |
| ComponentNotInManifest1 | ⊗ | | | |
| EventOrdering1 | ◯ | ✓ | ✓ | ✓ |
| IntentSink2 | ✓ | ✓ | ✓ | ✓ |
| SharedPreferences1 | ◯ | ✓ | ✓ | ✓ |
| Singletons1 | ◯ | ✓ | ✓ | ✓ |
| UnresolvableIntent1 | ✓ | ✓ | ✓ | ✓ |

✓= Correct warning, ⊗ = False warning, ◯ = Missed leak,

All-empty row = no leaks expected

**جدول ۲.** نتیجه مقایسه ابزارها بر اساس معیار DroidBench2

| | FD | DPF | DS | رویکرد ما |
|---|---|---|---|---|
| Precision | ۸۵٪ | ۱۰۰٪ | ۸۹٪ | ۱۰۰٪ |
| recall | ۸۶٪ | ۸۷٪ | ۹۲٪ | ۹۸٪ |
| F-measure | ۸۵٪ | ۹۳٪ | ۹۰٪ | ۹۹٪ |

**جدول ۳.** مقایسه ابزار با سایر رویکردها بر اساس معیار ICCBench

| | Amandroid | DroidSafe | رویکرد ما |
|---|---|---|---|
| Precision | ۱۰۰٪ | ۱۰۰٪ | ۱۰۰٪ |
| recall | ۱۰۰٪ | ۷/۱٪ | ۱۰۰٪ |
| F-measure | ۱۰۰٪ | ۱/۷۵٪ | ۱۰۰٪ |

انعکاسی برمی‌گردد.

## ۶ نتیجه‌گیری

در این مقاله به ارائه راه‌کاری جهت بررسی ویژگی نشت اطلاعات با استفاده از JPF پرداخته شده است. از آنجایی که JPF با بایت‌کد کار می‌کند، ابزار مهندسی معکوس اندروگارد به کار رفته است. در رویکرد پیشنهادی به منظور مدل‌سازی و افزایش مقیاس‌پذیری JPF از چارچوب‌های JPA و OCSEGEN (به ترتیب) استفاده می‌کند. همچنین برای تعریف ویژگی از امکان شنونده در JPF استفاده شده است. مزیت اصلی روش پیشنهادی

به ویژگی‌های خاص و منحصر به فرد بررسی‌کننده مدل JPF برمی‌گردد. قدرت این بررسی‌کننده مدل در تحلیل برنامه‌های همروند، تحلیل چندنخی و قدرت تحلیل متدهای انعکاسی باعث شده که روش ارائه‌شده در این مقاله با دقت بیشتری نسبت به کارهای مشابه که ابزارهای معروف و پرقدرتی در این زمینه هستند بتواند نشت اطلاعات در برنامه‌های موجود در محک معیار DroidBench و ICCBench را تشخیص دهد.

ما در ادامه این تحقیق به دنبال ارائه روش پیاده‌سازی شده در قالب یک ابزار کاربردی برای تشخیص نشت اطلاعات در برنامه‌های آندرویدی هستیم.

## مراجع


[1] Andrew Ahn. How we fought bad apps and malicious developers in 2017, 2018.

[2] Felivel Camilo, Andrew Meneely, and Meiyappan Nagappan. Do bugs foreshadow vulnerabilities? a study of the chromium project. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, pages 269–279. IEEE, 2015.

[3] Positive Technologies. Vulnerabilities and threats in mobile applications, 2019. Mobile Application Security Threats and Vulnerabilities 2019: Mobile Device Security - Attacks Research, 11-Sep 2019. [Accessed: 03-Apr-2020].

[4] Yepang Liu, Chang Xu, Shing-Chi Cheung, and Jian Lü. Greendroid: Automated diagnosis of energy inefficiency for smartphone applications. *IEEE Transactions on Software Engineering*, 40(9):911–940, 2014.

[5] Heila Botha. JPF-Android. Available: https://bitbucket.org/heila/jpf-android/src/master. [Accessed: 22-Apr-2020].

[6] H. V. D. Merwe. *Verifying Android Applications Using Java PathFinder*. PhD thesis, 2008.

[7] Oksana Tkachuk. *Domain-specific environment generation for modular software model checking*. PhD thesis, Kansas State University, 2008.

[8] Patrick Lam, Eric Bodden, Ondrej Lhoták, and Laurie Hendren. The soot framework for java program analysis: a retrospective. In *Cetus Users and Compiler Infastructure Workshop (CETUS 2011)*, volume 15, 2011.

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

[10] Mohsin Junaid, Donggang Liu, and David Kung. Dexteroid: Detecting malicious behaviors in android apps using reverse-engineered life cycle models. *computers & security*, 59:92–117, 2016.

[11] William Enck, Peter Gilbert, Seungyeop Han, Vasant Tendulkar, Byung-Gon Chun, Landon P Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N Sheth. Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. *ACM Transactions on Computer Systems (TOCS)*, 32(2):1–29, 2014.

[12] Guangdong Bai, Quanqi Ye, Yongzheng Wu, Heila Botha, Jun Sun, Yang Liu, Jin Song Dong, and Willem Visser. Towards model checking android applications. *IEEE Transactions on Software Engineering*, 44(6):595–612, 2017.

[13] Zohreh Bohluli and Hamid Reza Shahriari. Detecting privacy leaks in android apps using inter-component information flow control analysis. In *2018 15th International ISC (Iranian Society of Cryptology) Conference on Information Security and Cryptology (ISCISC)*, pages 1–6. IEEE, 2018.

[14] Zipeng Zhang and Xinyu Feng. Androidleaker: A hybrid checker for collusive leak in android applications. In *International Symposium on Dependable Software Engineering: Theories, Tools, and Applications*, pages 164–180. Springer, 2017.

[15] Mu Zhang and Heng Yin. *Android Application Security: A Semantics and Context-Aware Approach*. Springer, 2016.

[16] Siegfried Rasthofer, Steven Arzt, and Eric Bodden. A machine-learning approach for classifying and categorizing android sources and sinks. In *NDSS*, volume 14, page 1125, 2014.

[17] Secure-Software-Engineering. Droidbench. Available: https://github.com/secure-software-engineering/DroidBench. [Accessed:07-Jun-2020].

[18] Nastaran Shafiei. *Model Checking of Distributed Multi-Threaded Java Applications*. PhD thesis, 2014.

[19] Javapathfinder. javapathfinder/jpf-core. [Accessed: 22-Apr-2020].

[20] Areej Alzaidi, Suhair Alshehri, and Seyed M Buhari. Droidrista: a highly precise static data flow analysis framework for android applications. *International Journal of Information Security*, 19(5):523–536, 2020.

[21] Zi-Peng Zhang, Ming Fu, and Xin-Yu Feng. A lightweight dynamic enforcement of privacy protection for android. *Journal of Computer Science and Technology*, 34(4):901–923, 2019.