Journal of Electronical & Cyber Defence
Vol. 9, No. 2, 2021, Serial No. 34

PIVATool– A Fast and Precise Tool for Analysis and Detection of PendingIntent Vulnerabilities

A. Sarvazimi, M. Sakhaei-nia *

*Computer Engineering Department, Engineering Faculty,, Bu-Ali Sina University

(Received: 01/08/2020, Accepted: 26/10/2020)

ABSTRACT

Inter-component communication capability and specifically the PendingIntents have expanded the development of android applications. Although PendingIntent is used in many android applications, its improper use carries risks and can lead to various attacks such as denial of service, privilege escalation and data leakage. Therefore, it is important to detect vulnerabilities associated with PendingIntent before android apps are published by Android app stores. One of the challenges of analyzing and detecting vulnerabilities for Android markets is the running time duration of the vulnerability detection tools. In this paper, a new method has been proposed to detect vulnerabilities associated with PendingIntent. PIVATool is a tool based on static analysis for detecting PendingIntent-related vulnerabilities that takes less time to detect vulnerabilities without compromising precision. For evaluation, PIVATool is compared with the PIAnalyzer tool. The results on 51 selected program benchmarks showed that on average, PIVATool detects vulnerabilities 27% faster than PIAnalyzer with the same precision.

Keywords: PendingIntent, vulnerability, PIVATool, PIAnalyzer

* Corresponding Author Email: sakhaei@basu.ac.ir

نشریه علمی "پدافند الکترونیکی و سایبری"
سال نهم، شماره ۲، تابستان ۱۴۰۰، ص ۸۳-۷۵

علمی – پژوهشی

PIVATool – ابزار سریع و دقیق برای تحلیل و شناسایی آسیب‌پذیری‌های PendingIntent

آزاده سروعظیمی[۱]، مهدی سخایی‌نیا[۲*]

۱- دانشجوی کارشناسی ارشد مهندسی کامپیوتر و ۲- استادیار گروه مهندسی کامپیوتر، دانشکده فنی و مهندسی، دانشگاه بوعلی سینا، همدان، ایران
(دریافت: ۱۳۹۹/۰۵/۱۱، پذیرش: ۱۳۹۹/۰۸/۵)

چکیده

قابلیت ارتباط بین مؤلفه‌ای و به‌طور مشخص PendingIntentها سبب شده توسعه برنامه‌های کاربردی تحت اندروید گسترش یابد. گرچه PendingIntent در بسیاری از برنامه‌ها کاربرد دارد اما استفاده نادرست از آن مخاطراتی را به همراه داشته و می‌تواند زمینه حملات مختلفی مانند رد خدمت، ترفیع امتیاز و نشت داده‌ها را فراهم کند. بنابراین شناسایی آسیب‌پذیری‌های مرتبط با PendingIntent قبل از انتشار برنامه‌ها توسط فروشگاه‌های ارائه‌دهنده برنامه‌های اندروید اهمیت دارد. یکی از چالش‌های تحلیل و شناسایی آسیب‌پذیری‌ها برای بازارهای اندروید مدت‌زمان اجرای برنامه شناسایی کننده آسیب‌پذیری است. در این مقاله یک روش نوین برای شناسایی آسیب‌پذیری‌های مرتبط با PendingIntent ارائه گردیده است. PIVATool یک ابزار مبتنی بر تحلیل ایستا برای شناسایی آسیب‌پذیری‌های مرتبط با PendingIntent است که برای شناسایی آسیب‌پذیری‌ها به زمان کمتری نیاز داشته، بدون این‌که دقت کاهش پیدا کند. برای ارزیابی، PIVATool با ابزار PIAnalyzer مقایسه گردیده است. نتایج ارزیابی بر روی ۵۱ برنامه منتخب نشان داد که PIVATool به‌طور متوسط ۲۷ درصد سریع‌تر از PIAnalyzer آسیب‌پذیری‌ها را با همان دقت شناسایی می‌نماید.

کلیدواژه‌ها: PendingIntent، آسیب‌پذیری، PIVATool، PIAnalyzer

۱- مقدمه

با افزایش روزافزون محبوبیت و تعداد کاربران سیستم‌عامل اندروید، هزاران برنامه کاربردی هرروز در بازار رسمی اندروید (Google Play) و همچنین برخی از بازارهای جایگزین ظاهر می‌شوند. این محبوبیت و گسترش به دلیل قابلیت‌هایی است که در توسعه برنامه‌های کاربردی تحت اندروید ارائه می‌گردد [۱]. اما بعضاً این قابلیت‌ها آسیب‌پذیری‌هایی را به همراه دارد [۲] که سبب شده از آن سوءاستفاده گردد. یکی از این قابلیت‌ها ارتباط بین مؤلفه‌ای و مشخصاً PendingIntent است [۳]. این قابلیت در برنامه‌های کاربردی تحت اندروید بسیار کاربرد دارد مانند اعلان‌ها و هشدارها، اما استفاده نادرست از آن مخاطرات مختلفی برای استفاده‌کننده داشته و ممکن است منجر به حملاتی مانند رد خدمت، ترفیع امتیاز و یا نشت داده‌ها گردد [۳ و ۴]. عدم شناسایی این آسیب‌پذیری‌ها در برنامه‌های اندروید سبب می‌گردد که قربانی حملات این آسیب‌پذیری‌ها، پس‌ازآن کمتر از برنامه های کاربردی اندروید استفاده نموده و ضرر و زیان این صنعت را در پی خواهد داشت. بنابراین ضرورت دارد که این آسیب‌پذیری‌ها قبل از انتشار برنامه کاربردی در بازارهای مختلف ارائه‌دهنده برنامه‌های کاربردی مبتنی بر اندروید شناسایی گردد. اما بررسی برنامه‌های ارائه‌شده در بازارهای مختلف برای فروشگاه‌های ارائه‌کننده دارای چالش‌های گوناگونی ازجمله زمان‌بر بودن بررسی و تحلیل است.

PIها [۱] یک ویژگی قدرتمند اندروید برای ارتباطات بین مؤلفه‌ای هستند. PI یک Intent پایه را که قرار است بعداً توسط یک مؤلفه یا برنامه دیگر اما با هویت و مجوزهای برنامه فرستنده اجرا شود، ذخیره می‌کند[۴]. درواقع PI نشانه‌ای است که یک برنامه به‌عنوان‌مثال *برنامه الف* به برنامه دیگری مانند *برنامه ب* می‌دهد و با این کار به *برنامه ب* اجازه می‌دهد تا مجوزهای *برنامه الف* را برای اجرای یک قطعه کد از پیش تعریف‌شده به ارث ببرد. از همین رو، استفاده ناامن از PIها می‌تواند منجر به عواقب شدید امنیتی در قالب حملات رد خدمت[۲]، سرقت هویت[۳] و ترفیع[۴] امتیاز شود [۵ و ۶].

رویکردهای متعددی برای شناسایی آسیب‌پذیری‌های PI

[۱] برای سادگی در ادامه مقاله از خلاصه‌شده PendingIntent یعنی PI استفاده می‌گردد.
[۲] Denial of Service(DoS)
[۳] Identity theft
[۴] Privilege escalation

* رایانامه نویسنده مسئول: Sakhaei@basu.ac.ir

ناشر: دانشگاه جامع امام حسین (ع)

پیشنهادشده‌اند که در دقت، زمان اجرا و نوع تحلیل متفاوت هستند. XmanDroid یک چارچوب امنیتی است که محدودیت دقت داشته و تعداد زیادی هشدار کاذب دارد [۶]. مقاله [۷] یک رویکرد مبتنی بر ایستاست که قادر است تنها یک مورد از آسیب‌پذیری‌های PI را شناسایی کرده و از طرفی محدودیت دقت پایین و مثبت کاذب زیادی دارد. IntentDroid [۸] یک رویکرد مبتنی بر تحلیل پویا بوده که محدودیت‌های تحلیل پویا مانند سربار زمان اجرا را به ارث برده است. PIAnalyzer [۹] یک رویکرد مبتنی بر تحلیل ایستا است که در مقایسه با سایر رویکردها قادر است تعداد آسیب‌پذیری‌های مرتبط با PI بیشتری را با دقت بالا شناسایی نماید و اما چالش مدت‌زمان اجرا را دارد.

در این مقاله PIVATool[۱]، یک ابزار سریع و دقیق، مبتنی بر تحلیل ایستا برای شناسایی آسیب‌پذیری‌های مرتبط با PI، ارائه گردیده است. روش ارائه‌شده برای شناسایی آسیب‌پذیری‌ها به زمان کمتری نیاز داشته، بدون اینکه دقت کاهش پیدا کند. برای ارزیابی، PIVATool با ابزار PIAnalyzer مقایسه گردیده است. به این منظور هر دو ابزار پیاده‌سازی گردیده و به‌صورت تجربی با مجموعه‌ای از ۵۱ برنامه منتخب ارزیابی‌شده است. ارزیابی‌ها نشان می‌دهد که PIVATool به‌طور متوسط ۲۷ درصد سریع‌تر از PIAnalyzer آسیب‌پذیری‌ها را شناسایی می‌نماید.

عمده مشارکت‌های این پژوهش عبارت‌اند از:

- ارائه یک روش جدید برای شناسایی آسیب‌پذیری‌های مرتبط با PI که مبتنی بر تحلیل ایستا بوده و در مقایسه با رویکردهای ذکرشده سریع‌تر عمل می‌کند.
- توسعه PIVATool به‌عنوان یک ابزار منبع باز[۲] برای تحلیل آسیب‌پذیری‌های مرتبط با PI.
- ارزیابی دو رویکرد PIVATool و PIAnalyzer بر روی ۵۱ برنامه که به‌صورت تصادفی از فروشگاه Google Play انتخاب‌شده‌اند.

ساختار ادامه مقاله به شرح زیر است: در بخش دوم مفاهیم پایه مرتبط با مؤلفه‌های برنامه‌های کاربردی اندروید و اشیا Intent و PI که برای ارتباط بین مؤلفه‌ها مورداستفاده قرار می‌گیرند و همچنین آسیب‌پذیری‌های مرتبط با PI معرفی می‌شود. در بخش سوم کارهای مرتبط تشریح می‌گردد. بخش چهارم به شرح تفصیلی روش پیشنهادی جهت تحلیل و شناسایی آسیب‌پذیری‌های PI اختصاص دارد. در بخش پنجم ارزیابی روش پیشنهادی و مقایسه با PIAnalyzer ارائه‌شده است و در بخش ششم با نتیجه‌گیری مقاله پایان می‌یابد.

[۱] Pending Intent Vulnerability Analysis Tool(PIVATool)
[۲] Https://github.com/msniea/PIVATool

۲- مفاهیم پایه

در این بخش مفاهیم پایه که برای مطالعه بخش‌های بعدی لازم است، مرور می‌گردد. یک برنامه اندروید شامل چندین فایل و پوشه است که یکی از فایل‌های کلیدی در ساختار برنامه‌های اندروید فایل مانیفست، AndroidManifest.xml است. فایل مانیفست اندروید شامل فهرستی از مؤلفه‌هایی است که توسط یک برنامه میزبانی می‌شوند. برخی از اطلاعات آن برای ارتباط مؤلفه‌ها در زمان اجرا مورداستفاده قرار می‌گیرند و برخی دیگر مربوط به مجوزهای اعمال‌شده و درخواست‌شده توسط برنامه هستند [۱۰].

هر برنامه اندروید از واحدهای اساسی به نام مؤلفه ساخته‌شده است. چهار نوع مؤلفه وجود دارد: (۱) فعالیت‌هایی[۳] که رابط کاربری را نشان می‌دهند و قسمت قابل‌ملاحظه‌ای از برنامه‌های اندروید را تشکیل می‌دهند، (۲) گیرنده پخش[۴] که منتظر دریافت پیام‌هایی از سیستم اندروید و یا سایر برنامه‌های اندروید هستند مانند تماس‌های دریافتی یا پیام‌های متنی از سایر مؤلفه‌های سیستم یا سامانه‌های دیگر، (۳) ارائه‌دهنده محتوا[۵] که به‌عنوان رابط استاندارد برای به اشتراک گذاشتن داده‌ها بین برنامه‌ها عمل می‌کند و (۴) خدمت[۶] که برنامه‌ها را در پس‌زمینه اجرا می‌کند. مؤلفه‌های خدمت منحصربه‌فرد هستند زیرا پردازش آن‌ها در دستگاه کاربر پنهان است و فرصت‌های زیادی برای اقدامات مخرب ایجاد می‌کنند [۱۱].

در برقراری ارتباط بین مؤلفه‌های اندروید، برای مشخص کردن مؤلفه مقصد یک پیام، از Intent استفاده می‌گردد. دو نوع Intent وجود دارد: Intent صریح و Intent ضمنی. درصورتی‌که در یک Intent مؤلفه مقصد با نام آن مشخص شود، Intent صریح است و بایستی به همان مؤلفه تحویل داده شود. اما اگر در Intent به‌جای یک نام یک action مشخص شود، Intent ضمنی است و می‌تواند درخواست را به هر برنامه‌ای که قابلیت انجام action مشخص‌شده را دارد و آن action را در فیلتر Intent خود اعلام نموده است، تحویل دهد. برای این‌که یک خدمت یا فعالیت یک Intent را دریافت کند، باید در مانیفست اعلام شوند (گیرنده‌های پخش می‌تواند در مانیفست یا در زمان اجرا اعلام شوند). اگر در مانیفست برای یک مؤلفه علامت EXPORTED تنظیم شود یا مؤلفه شامل حداقل یک فیلتر Intent باشد آن‌وقت آن مؤلفه عمومی است. بدین معنی که می‌تواند Intentها را از دیگر برنامه‌ها دریافت کند و فیلترهای Intent مشخص می‌کنند که چه نوع

[۳] Activity
[۴] Broadcast receiver
[۵] Content provider
[۶] Service

Intentای باید به آن تحویل داده شود [۱۲].

PI نوع خاصی از Intent است که یک Intent پایه را کـه قـرار است بعداً توسط یک مؤلفه یا برنامه دیگر اما با هویت و مجوزهای برنامه فرستنده اجرا شود، ذخیره می‌کند. یک برنامه می‌تواند یک PI را به برنامه شخص ثالث و یا مؤلفه‌های سیستمی ارسال کنـد تا وظایف از پیش تعریف‌شده را در زمان دیگری از طـرف برنامـه فرستنده انجام دهند. تنها نکته‌ای که وجـود دارد ایـن اسـت کـه نیازی نیست برنامه اصلی در آن زمان در حافظـه بـوده یـا فعـال باشد، زیرا گیرنده آن را به صورتی که توسط برنامـه اصـلی اجـرا می‌شود، اجرا خواهد کرد. برای این منظور، اندروید مجوز و هویت برنامه فرستنده را به برنامه مقصدی کـه PI را دریافـت مـی‌کنـد، منتقـل مـی‌کنـد. درواقـع PI، Intentای اسـت کـه action مشخص‌شده توسط برنامه ارسال‌کننده را در آینده و صرف‌نظـر از این‌که برنامه در حال اجرا است یا خیر به نمایندگی از آن برنامـه یعنی با هویت و مجوزهای آن، انجـام مـی‌دهـد [۴، ۱۳، ۱۴]. PI بیشترین استفاده را برای خدمات اطلاع‌رسانی و زنگ هشدار دارد [۳].

هر زمـان کـه گیرنـده PI، متـد Send آن را فراخـوانی کنـد، Intent پایه مرتبط با آن با هویت و مجوزهای برنامه‌ای کـه آن را ایجاد کرده است، اجرا می‌شود [۷]. بااین‌حال، گیرنده PI می‌تواند سه بخش اصلی از داده‌های Intent پایه را تغییر داده کـه ممکـن است معنای Intent پایه که با مجوز و هویـت برنامـه اصـلی اجـرا می‌گردد، تغییر نماید. اگـر مؤلفـه مقصـد یـا Intent action قـبلاً توسط فرستنده تعریف‌شده باشد، گیرنده PI دیگر نمی‌تواند آن را بازنویسی[۱] کند، اما اگر از قبل توسط فرستنده تعریف‌نشده باشـد ممکن است پس از انتقال آن بـه گیرنـده یـک مؤلفـه مقصـد یـا Intent action ناخواسته تعریف شود. درنهایت، داده‌های اضـافی[۲]، که درواقع یک انباره کلیـد-مقـدار[۳] هسـتند همیشـه مـی‌تواننـد توسط گیرنده PI پس از دریافت افزوده شوند. پیامدهای این امـر حاکی از آن است کـه یـک Intent ضـمنی(که مؤلفـه مقصـد آن تعریف‌نشده) می‌تواند توسط برنامه گیرنده تغییر یابد تا بتواند هر مؤلفه موردنظر خود را(با مجوز برنامـه اصـلی) هـدف قـرار دهـد، ازجمله ویژگی‌های سیستم مانند پاک کردن تلفن [۹]. پیامد این آسیب‌پذیری‌ها حملاتی است که منجـر بـه ترفیـع امتیـازات، رد خـدمت و نشـت داده‌هـا مـی‌گـردد. ازایـن‌رو شناسـایی ایـن آسیب‌پذیری‌ها دارای اهمیت زیادی است.

PI می‌تواند یا به یک مؤلفه سیستمی قابل‌اعتماد منتقل‌شده (مانند مدیر زنگ هشدار) و یا به Intent دیگری که آن را پوشانده است منتقل شود. آسیب‌پذیری‌های ناشی از PI بسـتگی بـه نـوع ارسال آن دارد که به‌طور مختصر در ادامه تشریح شده‌اند.

۲-۱- PI ضمنی ارسال‌شده به مؤلفه سیستمی

آسیب‌پذیری: برنامه X می‌تواند یک PI شامل یک Intent ضـمنی را ایجاد کرده و آن را به یک مؤلفه سیستمی مانند مدیر اعلان‌ها[۴] ارسال کند [۹].

سوءاستفاده: زمانی که کاربر بر روی اعلان کلیک می‌کنـد انتظـار می‌رود که مؤلفه مورد انتظار برنامه X، Intent را تحویل بگیرد. اما هر برنامه‌ای که قابلیت انجام action مشخص‌شده در Intent پایـه را دارد و آن action را در فیلتر Intent خود اعـلام نمـوده اسـت، می‌تواند Intent را تحویـل بگیـرد کـه در ایـن صـورت حملـه رد خدمت رخ می‌دهد.

۲-۲- PI ضمنی پوشانده شده در Intent ضمنی

آسیب‌پذیری: برنامه X می‌تواند یک PI شامل یک Intent ضـمنی را ایجاد کند و آن را از طریق یـک Intent ضـمنی ارسـال نمایـد. هنگامی‌که PI پردازش می‌شود، Intent پایه مرتبط با آن بر اساس فیلتر Intent، توسط یک مؤلفه شناسایی‌شده و پردازش می‌شـود. هنگامی‌که چندین مؤلفـه دارای فیلتـر Intent یکسـانی هسـتند، برای پـردازش Intent مربوطـه مؤلفـه بـا اولویـت بـالاتر انتخـاب می‌شود [۱۵].

سوءاستفاده: هر برنامه می‌تواند یک PI ضـمنی را دریافـت کـرده [۱۵] و از متد send آن PI بـرای ارسـال Intentهـا بـا داده‌هـای دلخواه از طرف فرستنده اولیـه اسـتفاده کنـد. درنتیجـه، برنامـه مخرب می‌تواند داده‌های Intent را دست‌کـاری کـرده و درنتیجـه نشت یا تغییر داده‌های حساس و بازپخش کردن PI می‌تواند برای حملات Intent spoofing مورداستفاده قـرار گیـرد و حمـلات رد خدمت و نشت اطلاعات رخ می‌دهد [۷].

۲-۳- PI خالی پوشانده شده در Intent ضمنی

آسیب‌پذیری: برنامه X می‌تواند یک PI شامل یک Intent خالی را ایجاد کرده و آن را از طریق یک Intent ضـمنی ارسـال نمایـد. هنگـامی‌کـه هـیچ actionای در PI مشـخص نشـود، گیرنـده PI مـی‌توانـد هرگونـه actionای را تنظـیم کـرده و آن را از طـرف برنامه‌ای که PI را ارسال کرده است، انجام دهد [۱۵].

سوءاستفاده: برنامه مخرب می‌تواند علاقه خود را از طریق فیلتر

[۱] overridden
[۲] Extra date
[۳] Key-value store
[۴] Notification Manager

Intent به PI نشان دهد. به‌محض دریافت PI خالی، برنامه مخرب یک action مخرب را در PI تنظیم می‌کند و هنگامی‌که PI پردازش شود، action مخرب از طرف برنامه X انجام خواهد شد [۱۵]. درنتیجه حملات ترفیع امتیاز و رد خدمت رخ می‌دهد.

۲-۴- یک مثال از PendingIntent آسیب‌پذیر

یک نمونه کد از استفاده ناامن از یک PendingIntent در شکل (۱) نشان داده شده است (نمونه کد در منبع [۹] ارائه شده است).

```
1 protected void onCreate(Bundle savedInstanceState) {
2   super.onCreate(savedInstanceState);
3   setContentView(R.layout.activity_main_vuln);
4   Intent baseIntent = new Intent();
5   PendingIntent pendingIntent = PendingIntent.getActivity(this, 1,
      baseIntent, PendingIntent.FLAG_UPDATE_CURRENT);
6   Intent implicitWrappingIntent = new Intent(Intent.ACTION_SEND);
7   implicitWrappingIntent.putExtra("vulnPI", pendingIntent);
8 sendBroadcast(implicitWrappingIntent);
9 }
```

الف- برنامه آسیب‌پذیر

```
1 public void onReceive(Context context, Intent intent) {
2   Bundle extras = intent.getExtras();
3   PendingIntent pendingIntent = (PendingIntent) extras.get("vulnPI");
4   Intent vunlnIntent = new Intent(Intent.ACTION_CALL, Uri.parse("tel:" +
        "0900123456789"));
5   try {
6   pendingIntent.send(context, 2, vunlnIntent, null, null);
7   } catch (PendingIntent.CanceledException e) { e.printStackTrace(); }
8 }
```

ب - برنامه مخرب.

شکل (۱): نمونه کد از یک برنامه حاوی PendingIntent آسیب‌پذیر [۹]

در کد ارائه‌شده در شکل (۱) بخش (الف)، یک نمونه برنامه آسیب‌پذیر نشان داده‌شده است. این برنامه مجوز برقراری تماس تلفنی را دارد. در خط ۴ این برنامه، یک Intent پایه خالی ایجاد شده است. در خط ۵ این Intent پایه در داخل PendingIntent پوشانده شده و از طریق یک Intent ضمنی دیگر به‌نام implicitWrappingIntent ارسال شده است (خطوط ۶ تا ۸). از آنجا که Intent حاوی PendingIntent در برنامه آسیب‌پذیر به‌صورت ضمنی ارسال می‌گردد، می‌تواند توسط هر برنامه‌ای که فیلتر متناظر با آن‌را در مانیفست خود اعلام نموده است، دریافت گردد. قطعه کد ارائه‌شده در بخش (ب) شکل (۱)، نشان‌دهنده یک برنامه مخرب است که می‌تواند بر اساس فیلتر Intent خود، Intent ضمنی ارسال‌شده توسط برنامه آسیب‌پذیر را دریافت نموده و PendingIntent پوشانده‌شده در آن را دریافت نماید. این برنامه مجوز برقراری تماس تلفنی را ندارد. در خط ۳ این کد، PendingIntent استخراج شده و با توجه به اینکه Intent پایه در کد آسیب‌پذیر یک Intent خالی بوده و هیچ actionای برای آن تعریف‌نشده است، در خط ۴ Intent پایه دست‌کاری شده و یک action به‌منظور برقراری تماس تلفنی با یک شماره تلفن مشخص تعریف‌شده است. برنامه مخرب در خط ۶ با فراخونی متد send شیء PendingIntent توانسته است با هویت و مجوز برنامه آسیب‌پذیر تماس تلفنی برقرار نماید. همان‌طور که تشریح گردید، برنامه مخرب توانست با سوءاستفاده از برنامه آسیب‌پذیر که به‌صورت ناامن از PendingIntent استفاده نموده است، موفق به اجرای خواسته مخرب خود گردد.

۳- کارهای مرتبط

رویکردهای متعددی برای تحلیل و شناسایی آسیب‌پذیری‌های PI وجود دارد که در نوع تحلیل، دقت، زمان اجرا و کارایی متفاوت هستند. رویکردهای امنیتی موجود به شیوه‌ها و معماری‌های مختلفی پیاده‌سازی شده و از فن‌ها و سازوکارهای مختلفی استفاده می‌کنند که به‌طورکلی به سه گروه اصلی تقسیم می‌شوند: مبتنی بر پیش‌گیری، مبتنی بر تحلیل (ایستا یا پویا)، نظارت در زمان اجرا [۱۰].

XManDroid امکان نظارت در زمان اجرای ارتباطات بین مؤلفه‌ها را فراهم می‌کند و سیاست‌های ارتباطی را بر اساس ترکیبی از مجوزهای خاص توسعه می‌دهد[۶]. ازآنجاکه XManDroid همه ارتباطات بین مؤلفه‌ها را رصد می‌کند، قادر است از بروز آسیب‌پذیری‌های PI جلوگیری کند. XManDroid در تمایز جریان خوش‌خیم ارتباطات بین مؤلفه‌ها از موارد تبانی (حمله colluding) محدودیت‌هایی داشته و تعداد زیادی هشدار کاذب تولید می‌کند. این رویکرد بیشتر برای نظارت بر تعداد کمی از برنامه‌های نصب‌شده در یک دستگاه مناسب است.

روش ارائه‌شده در مقاله [۷] یک رویکرد مبتنی بر تحلیل ایستا است که قادر به شناسایی تنها یک مورد از آسیب‌پذیری‌های مرتبط با PI است.

IntentDroid رویکرد دیگری است که به‌صورت پویا برنامه‌های اندروید را تحلیل می‌کند و می‌تواند آسیب‌پذیری‌های

مرتبط با PI را نیز شناسایی کند[۸]. IntentDroid تنها قادر است آسیب‌پذیری‌هایی که ناشی از ارتباطات بین مؤلفه‌های فعالیت برنامه‌ها هستند را شناسایی کند و آسیب‌پذیری‌های ناشی از ارتباطات مؤلفه‌های خدمت،گیرنده پخش و ارائه‌دهنده محتوا را بررسی نمی‌کند.

IccTA یک ابزار منبع باز برای انجام تحلیل آلودگی مبتنی بر ارتباط بین مؤلفه است که می‌تواند نشت‌های حریم خصوصی را با ارائه یک نمودار جریان کنترلی از طریق instrumentation کد برنامه‌ها شناسایی کند [۱۶]. IccTA قادر به شناسایی آسیب‌پذیری‌های مرتبط با PI است[۱۷].

PIAnalyzer یک ابزار مبتنی بر تحلیل ایستا است که به‌منظور شناسایی آسیب‌پذیری‌های مرتبط با PI طراحی‌شده است[۹]. PIAnalyzer بایت کد استخراج‌شده از فایل Apk را با استفاده از ابزار ApkTool که یک ابزار مبتنی بر مهندسی معکوس است، به کد میانی Smali تبدیل می‌کند. توسعه‌دهندگان PIAnalyzer به‌منظور تحلیل کدهای میانی Smali، ابزار Smali Slicer را طراحی نموده‌اند که بخش اصلی تحلیل PIAnalyzer را انجام می‌دهد. PIAnalyzer با استفاده از Smali Slicer مجموعه‌ای از دستورات تأثیرگذار و تأثیرپذیر را باز می‌گرداند. بعد از تحلیل دستورات درصورتی‌که PI دارای Intent پایه ضمنی باشد و به یک برنامه شخص ثالث ارسال شود، آسیب‌پذیری را گزارش کرده و درصورتی‌که به یک مؤلفه سیستمی ارسال شود هشدار می‌دهد. PIAnalyzer با دقت حدود ۹۰٪ قادر است آسیب‌پذیری‌های مرتبط با PI را بیابد اما ازنظر سرعت تحلیل برنامه‌ها نیز محدودیت دارد.

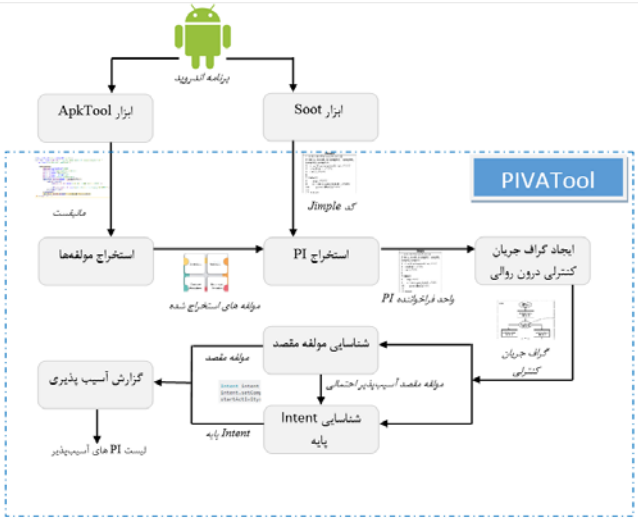
رویکرد ارائه‌شده در این مقاله در مقایسه با کارهای مرتبط بیان‌شده، قادر است آسیب‌پذیری‌های مرتبط با PI را با سرعت بیشتر و دقت مشابه شناسایی نماید.

۴- روش پیشنهادی

شکل (۲) نمای کلی PIVATool را نشان می‌دهد که در سه مرحله اصلی تحلیل صورت می‌گیرد. در مرحله اول PI در برنامه شناسایی می‌گردد. به این منظور با استفاده از ابزار ApkTool فایل مانیفست برنامه استخراج می‌گردد. استخراج مؤلفه‌ها در PIVATool بر اساس مانیفست مؤلفه‌هایی از برنامه که بایستی تحلیل شوند، شناسایی می‌نماید. از سوی دیگر با استفاده از ابزار Soot بایت‌کدهای Dalvik به کد سه آدرسه Jimple که یک نوع کد میانی Soot است، تبدیل می‌گردد. در مؤلفه استخراج PI با دریافت کد Jimple مؤلفه‌های استخراج‌شده از مرحله قبل، PIهای فراخوانی شده در هر مؤلفه را استخراج می‌کند.

در مرحله دوم تحلیل PI صورت می‌گیرد تا Intent پایه و مؤلفه مقصد PI شناسایی و وضعیت آن مشخص شود. برای این کار با ایجاد گراف جریان کنترلی درون روالی، بلاک‌های پایه حاوی PI و بلاک‌های قبلی و بعدی آن شناسایی می‌گردد تا بر اساس آن‌ها مؤلفه‌های مقصد PI و Intent پایه مرتبط به PI شناسایی گردد. Intent پایه در صورتی شناسایی می‌گردد که مؤلفه‌های مقصد احتمال آسیب‌پذیری آن وجود داشته باشد.

در مرحله آخر تحلیل و گزارش آسیب‌پذیری، درصورتی‌که احتمال آسیب‌پذیر بودن PI وجود داشت Intent پایه آن تحلیل می‌شود و در صورت وجود آسیب‌پذیری، نوع آن گزارش می‌گردد. در ادامه جزئیات هر مرحله تشریح می‌گردد.



شکل (۲): نمای کلی PIVATool.

۴-۱- شناسایی PI

فایل برنامه کاربردی اندروید، Apk، با استفاده از ابزار ApkTool، مهندسی معکوس شده و فایل مانیفست آن استخراج می‌گردد. با استفاده از فایل مانیفست برنامه می‌توان مؤلفه‌های برنامه را

شناسایی و استخراج نمود. برای شناسایی PI نیاز است که مؤلفه فعالیت، گیرنده پخش و خدمت تحلیل گردد. بنابراین PIVATool بخش‌هایی از کد Jimple که به مؤلفه‌های موردنظر اختصاص دارد را تحلیل می‌کند. در بستر اندروید، کد منبع جاوا به فایل‌های class.کامپایل می‌شوند و پس‌ازآن با استفاده از ابزار "dx"به فایل dex. تبدیل می‌شوند. این فایل dex. حاوی بایت‌کد Dalvik است که در ماشین مجازی Dalvik اجرا می‌شود. PIVATool این بایت‌کدهای Dalvik را با استفاده از ابزار Soot به کد میانی Jimple تبدیل می‌کند. کد میانی Jimple یک کد سه آدرسه از بایت‌کد است که قابلیت تحلیل ایستای برنامه‌های اندروید را بهبود می‌بخشد. یک نمونه کد جاوا و ترجمه‌شده آن به کد میانی Jimple توسط Soot در شکل (۳) نمایش داده‌شده است. زنجیره[1] یک ساختار داده collection است که دسترسی بازمان ثابت را به عناصر آن فراهم می‌کند. در بدنه کد Jimple سه نوع زنجیره وجود دارد. زنجیره متغیرهای محلی[2]، زنجیره واحد یا دستورات[3] و زنجیره استثنائات[4]. در زنجیره متغیرهای محلی، متغیرهایی در برنامه هستند که یا در یک دستور مقداری را می‌پذیرند[5] و یا در یک دستور از مقدار آن‌ها استفاده می‌شوند[6]. همان‌طور که در شکل (۳) مشاهده می‌شود در قسمت بالای متد زنجیره متغیرهای محلی همراه با نوع مقداری که می‌پذیرند، تعریف می‌شوند. زنجیره واحدها شامل دستورات اصلی برنامه هستند و زنجیره استثنائات شامل استثناهای برنامه هستند.

برای شناسایی متغیرهای نوع PI در کد Jimple بخش زنجیره متغیرهای محلی بررسی‌شده و متغیرها از نوع PI شناسایی می‌گردد. سپس با استفاده از زنجیره واحدها، دستوراتی که متغیرهای PI مشخص‌شده در مرحله قبل در این دستورات تعریف می‌شوند (مقداردهی می‌گردند) را می‌یابد. درصورتی‌که این دستورات یکی از چهار متد ()getActivity، ()getActivities، ()getBroadcast و ()getService را فراخوانی می‌نماید، این دستورات به‌عنوان فراخواننده PI مشخص‌شده و تحلیل بر اساس این دستورات ادامه پیدا خواهد نمود. بنابراین خروجی این مرحله فهرستی از دستوراتی است که متغیرهایی از نوع PI را مقداردهی نموده و یکی از چهار متد ذکرشده را فراخوانی می‌نماید.

۴-۲- تحلیل PI

همان‌طور که در بخش ۲ مقاله، در خصوص آسیب‌پذیری‌های ناشی از PI ذکر گردید بسته به نوع Intent پایه و مؤلفه مقصد PI ممکن است PI دارای آسیب‌پذیری باشد. بنابراین، باید Intent پایه و همچنین مؤلفه مقصد شناسایی و بررسی گردد. به این منظور نیاز است وابستگی‌های داده‌ای و کنترلی دستورات حاوی مقداردهی متغیرهای PI با دستورات استفاده‌کننده از آن مشخص گردد. برای یافتن این وابستگی‌های کنترلی از گراف جریان کنترلی درون روالی[7] استفاده گردیده و پس‌ازآن با پیمایش این گراف وابستگی‌های داده‌ای تحلیل می‌گردد. بنابراین نیاز است برای متدهای حاوی دستورات مرحله قبل گراف جریان کنترلی ایجاد گردد. به این منظور کد متد حاوی دستورات، تحلیل گردیده و با شناسایی بلاک‌های پایه آن گراف جریان کنترلی آن ایجادشده است. بنابراین بلاک پایه حاوی دستورات فراخواننده PI و بلاک‌های قبلی و بعدی این بلاک بر اساس گراف جریان کنترلی در دسترس خواهد بود.

شکل (۳): نمونه‌ای از کد میانی Jimple.

[1] Chain
[2] Chain of Locals
[3] Chain of Units
[4] Chain of Traps
[5] Definition
[6] Use

[7] Intra-procedural Control Flow Graph(CFG)

با داشتن بلاک‌های پایه قبلی و بعدی دستورات فراخواننده PI، با پیمایش این بلاک‌ها می‌توان بلاک حاوی Intent پایه و مؤلفه مقصد PI را پیدا نمود و با تحلیل آن مشخص نمود که آیا PI به‌صورت امن استفاده‌شده است یا نه. ابتدا وضعیت مؤلفه مقصد بررسی می‌گردد و درصورتی‌که احتمال آسیب‌پذیر بودن آن وجود داشت وضعیت Intent پایه بررسی می‌گردد. مؤلفه مقصد، مؤلفه‌ای است که متغیر حاوی PI به آن ارسال می‌گردد. به این منظور بلاک‌های بعدی بلاک حاوی دستورات فراخواننده PI بررسی می‌شوند. اگر متغیرهای استفاده‌شده در این دستورات شامل متغیرهای حاوی PI باشند، به‌عنوان مؤلفه مقصد در نظر گرفته می‌شوند. درصورتی‌که دستورات متغیر حاوی PI را به یک مؤلفه سیستمی ارسال کنند و یا در داخل یک Intent ضمنی به‌عنوان داده‌های اضافی قرار دهند، احتمال آسیب‌پذیری برای آن وجود خواهد داشت از همین رو Intent پایه برای شناسایی و گزارش آسیب‌پذیری‌ها باید تحلیل گردد.

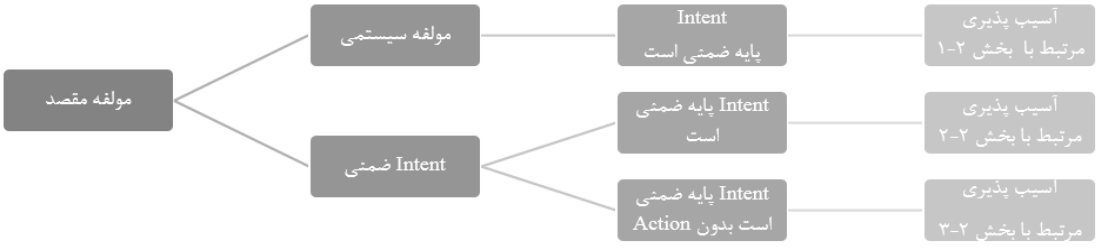
برای تحلیل Intent پایه ابتدا متغیرهایی که حاوی Intent پایه است، مشخص می‌گردد. سپس دستورات بلاک پایه‌ای که حاوی فراخواننده PI است و قبل از دستورات فراخواننده PI قرارگرفته‌اند، به‌منظور شناسایی نوع Intent پایه تحلیل می‌گردند. اگر دستورات مورد تحلیل، متد سازنده Intent پایه را فراخوانی کند و مؤلفه مقصد را به‌صورت صریح در پارامتر متد سازنده اعلام کرده باشد نوع Intent پایه صریح است. همچنین اگر دستورات مورد تحلیل یکی از پنج متد setClassName()، setClass()، setComponent()، setPackage()و setSelector() را فراخوانی کنند Intent پایه صریح در نظر گرفته می‌شود و در غیر این صورت نوع Intent پایه ضمنی خواهد بود. درصورتی‌که نوع Intent پایه در بلاک پایه حاوی دستور فراخواننده PI مشخص نگردید، دستوراتی که در بلاک‌های پایه قبل از آن قرارگرفته‌اند نیز تحلیل می‌شوند.

۴-۳- تحلیل و گزارش آسیب‌پذیری

مطابق دسته‌بندی ارائه‌شده در شکل (۴) و آسیب‌پذیری‌های مرتبط به PI در بخش دوم مقاله، درصورتی‌که PI به یک مؤلفه سیستمی (مؤلفه مقصد) ارسال‌شده باشد و Intent پایه آن ضمنی باشد، آسیب‌پذیری که در بخش ۲-۱ بیان شد، گزارش می‌شود. درصورتی‌که PI در داخل یک Intent ضمنی دیگر به‌عنوان داده‌های اضافی قرارگرفته باشد و Intent پایه آن ضمنی باشد آسیب‌پذیری تشریح شده در بخش ۲-۲ گزارش می‌شود. اما اگر Intent پایه آن ضمنی باشد و هیچ‌گونه Actionای برای آن تنظیم‌نشده باشد آسیب‌پذیری تشریح شده در بخش ۲-۳ گزارش می‌گردد.

۵- ارزیابی و نتایج

در این بخش PIVATool با رویکرد PIAnalyzer، ابزار منبع باز برای تحلیل آسیب‌پذیری‌های مرتبط با PI، مقایسه می‌گردد. برای مقایسه و ارزیابی بهتر نتایج، هر دو رویکرد ابتدا با استفاده از ابزار Soot به کد میانی Jimple تبدیل و سپس الگوریتم مرتبط با هر رویکرد پیاده‌سازی شده است. در ادامه زمان اجرای هر دو ابزار فوق با اجرای آن‌ها بر روی ۵۱ برنامه محک که به‌صورت تصادفی از Google Play انتخاب‌شده بودند، ارزیابی می‌گردد. تمام ارزیابی‌های صورت گرفته در این بخش بر روی یک سیستم با پردازنده Intel Core i3 با 4 GB حافظه RAM انجام‌شده است.



شکل (۴): دسته‌بندی تحلیل آسیب‌پذیری PI بر اساس مؤلفه مقصد و Intent پایه.

جدول (۱) نتایج حاصل از مقایسه زمان اجرای ابزارهای فوق در تشخیص آسیب‌پذیری‌های مرتبط با PI و شکل (۵) نمودار مرتبط با نتایج ارزیابی را ارائه می‌دهد. همه ۵۱ مورد آزمون همراه با اندازه برنامه و نتیجه زمان اجرای مربوط به هر ابزار در جدول (۱) ذکرشده است. دقت هر دو ابزار باهم یکسان بودند و آسیب‌پذیری‌ها را به‌صورت مشابه شناسایی می‌نمودند.

برای هر برنامه مورد تحلیل یکی از سه حالت زیر صدق می‌کند:

۱. برنامه شامل هیچ PI نیست.
۲. برنامه مورد تحلیل شامل یک یا چند PI آسیب‌پذیر است.
۳. برنامه مورد تحلیل شامل یک یا چند PI غیر آسیب‌پذیر است.

در حالت (۱)، PIVATool به این دلیل که فقط متغیرهای Local موجود در هر متد را جستجو می‌کند سریع‌تر متوجه عدم وجود PI می‌شود. اما PIAnalyzer به این دلیل که همه دستورات موجود در کد برنامه را جستجو می‌کند مدت زمان بیشتری نیاز

دارد تا متوجه عدم وجود PI در برنامه شود. بنابراین در حالت (۱)، PIVATool تحلیل سریع‌تری را انجام می‌دهد.

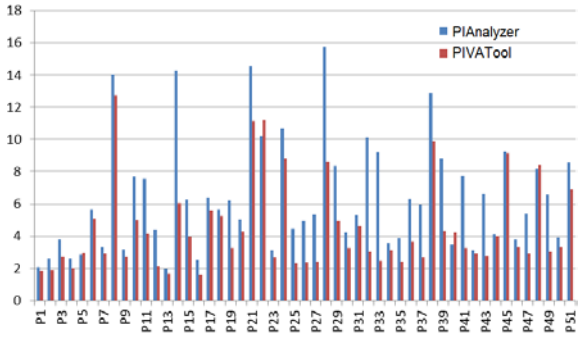
جدول (۱): نتایج زمان اجرای ابزارها بر روی برنامه‌های محک.

برنامه	اندازه برنامه (MB)	زمان اجرا (ثانیه) PIAnalyzer	زمان اجرا (ثانیه) PIVATool	درصد کاهش/ افزایش
P1	۱/۲۵	۲/۰۸	۱/۸۶	۱۱
P2	۰/۱۸۱	۲/۵۹	۱/۸۹	۲۷
P3	۲/۰۹	۳/۷۹	۲/۷۴	۲۸
P4	۰/۱۸۲	۲/۶۰	۲/۰۱	۲۳
P5	۲/۷۲	۲/۸۷	۲/۹۶	-۳
P6	۲/۳۳	۵/۶۳	۵/۰۸	۱۰
P7	۱/۳	۳/۳۰	۲/۹۲	۱۲
P8	۹/۴۶	۱۴	۱۲/۷۲	۹
P9	۰/۰۵	۳/۱۶	۲/۷۳	۱۴
P10	۱۵/۷	۷/۷۰	۵	۳۵
P11	۴/۷۲	۷/۵۷	۴/۱۴	۴۵
P12	۲۴	۴/۴۰	۲/۱۲	۵۲
P13	۱۳	۱/۹۶	۱/۶۶	۱۵
P14	۷/۰۸	۱۴/۳۰	۶/۰۳	۵۸
P15	۱۵/۲	۶/۲۵	۳/۹۸	۳۶
P16	۲/۴۸	۲/۵۱	۱/۶۴	۳۵
P17	۵/۴۲	۶/۳۹	۵/۶۲	۱۲
P18	۵/۵۸	۵/۶۴	۵/۲۵	۷
P19	۱۶/۹	۶/۲۳	۳/۲۴	۴۸
P20	۱/۲۵	۵/۰۱	۴/۲۶	۱۵
P21	۶/۵۷	۱۴/۵۶	۱۱/۱۵	۲۳
P22	۵/۷۳	۱۰/۲۳	۱۱/۱۸	-۹
P23	۱۱/۶	۳/۱۳	۲/۶۸	۱۴
P24	۱۴/۳	۱۰/۶۸	۸/۷۹	۱۸
P25	۱۰/۶	۴/۴۷۲	۲/۳۲	۴۸
P26	۲۳/۶	۴/۹۲	۲/۳۷	۵۲
P27	۷/۱۱	۵/۳۳	۲/۴۳	۵۴
P28	۱۶	۱۵/۷۴	۸/۶۰	۴۵
P29	۶/۸	۸/۳۴	۴/۹۶	۴۱
P30	۹/۱۱	۴/۲۵	۳/۲۸	۲۳
P31	۶۱/۵	۵/۲۸	۴/۶۴	۱۲
P32	۲۰/۴	۱۰/۱۱	۳/۰۵	۷۰
P33	۲۷/۲	۹/۲۰	۲/۴۷	۷۳
P34	۳/۱۷	۳/۵۷	۳/۱۴	۱۲
P35	۱۲/۴	۳/۸۷	۲/۴۱	۳۸
P36	۱۴/۳	۶/۳۳	۳/۶۷	۴۲
P37	۳/۱۸	۵/۹۴	۲/۶۹	۵۵
P38	۱۶/۲	۱۲/۸۶	۹/۹۲	۲۳
P39	۱۴/۵	۸/۷۸	۴/۳۰	۵۱
P40	۱۵/۶	۳/۵۰	۴/۲۵	-۲۱
P41	۲۵/۳	۷/۷۵	۳/۲۴	۵۸
P42	۱۱/۸	۳/۱۲	۲/۹۲	۶
P43	۱۸/۹	۶/۶۲	۲/۷۹	۵۸
P44	۹/۵۰	۴/۱۲	۴/۰۲	۲
P45	۹/۷۲	۹/۲۳	۹/۱۳	۱
P46	۱۵/۲	۳/۷۷	۳/۲۹	۱۳
P47	۳/۵	۵/۳۹	۲/۹۱	۴۶
P48	۶/۰۷	۸/۱۸	۸/۳۹	-۳
P49	۴/۵۵	۶/۵۹	۳/۰۲	۵۴
P50	۲/۹۴	۳/۹۱	۳/۳۲	۱۵
P51	۶/۰۸	۸/۵۶	۶/۸۹	۲۰
میانگین	۱۰/۶۷	۶/۴۰	۴/۴۳	۲۷

در حالت (۲)، PIVATool به این دلیل که برای انجام تحلیل فقط بلاک‌هایی که ممکن است حاوی مؤلفه مقصد PI و Intent پایه آن باشند را مورد تحلیل قرار می‌دهد و همه دستورات برنامه را بررسی نمی‌کند، آسیب‌پذیری مرتبط با PI را سریع‌تر شناسایی می‌کند.

حالت (۳)، مربوط به زمانی است که برنامه شامل PI صریح است و آن را به یک مؤلفه سیستمی ارسال می‌کند. در این حالت PIAnalyzer در مدت‌زمان کمتری متوجه عدم آسیب‌پذیری PI می‌شود، چراکه PIAnalyzer ابتدا نوع Intent پایه را تحلیل می‌کند و درصورتی‌که متوجه صریح بودن Intent پایه شود تحلیل را ادامه نمی‌دهد. اما PIVATool ابتدا مؤلفه مقصد PI را تحلیل کرده و سپس در مرحله بعد نوع Intent پایه را تحلیل می‌کند و به همین دلیل دیرتر متوجه عدم آسیب‌پذیر بودن PI می‌شود.

نتایج در جدول (۱) و شکل (۵) بیان‌گر این است که PIVATool در ۹۵٪ مواقع زمان اجرای بهتری نسبت به PIAnalyzer داشته است و در ۵٪ مواقع PIAnalyzer با اختلاف‌زمانی بسیار کمی سریع‌تر از PIVATool عمل می‌کند. نتایج ارزیابی‌های صورت گرفته نشان می‌دهد که PIVATool ازلحاظ سرعت تحلیل برنامه‌ها حدود 27٪ سریع‌تر از PIAnalyzer است.



شکل (۵): نمودار مقایسه زمان اجرای دو ابزار.

۶- نتیجه‌گیری

استفاده امن از PIها در توسعه برنامه‌های اندروید، باعث کاهش آسیب‌پذیری‌های برنامه‌های اندروید گردیده و جلو حملات مختلف مانند رد خدمت، نشت داده‌ها و ترفیع امتیاز گرفته می‌شود. چالش شناسایی این آسیب‌پذیری‌ها برای فروشگاه‌های ارائه‌دهنده این برنامه‌های کاربردی زمان تحلیل برنامه‌ها به‌منظور شناسایی آسیب‌پذیری‌ها است.

در این مقاله روشی ارائه گردید که آسیب‌پذیری‌های مرتبط با PI را در زمان کمتر و با دقت مشابه با سایر ابزارها شناسایی

می‌نمود. به این منظور کد برنامه کاربردی تحت اندروید به کد میانی Jimple تبدیل‌شده و از مؤلفه‌های آن PIها استخراج می‌گردد. سپس در PIVATool بر اساس وضعیت Intent پایه و مؤلفه مقصد آسیب‌پذیری‌های مختلف تحلیل و گزارش می‌گردد. برای شناسایی Intent پایه و مؤلفه مقصد تحلیل جریان داده بر روی گراف جریان کنترلی درون روالی متدهای حاوی فراخواننده PI انجام می‌شود. برای ارزیابی PIVATool این ابزار به همراه PIAnalyzer پیاده‌سازی گردید و بر روی ۵۱ برنامه منتخب از بازار برنامه‌های اندروید آزمون صورت گرفت. نتایج نشان داد که PIVATool ازلحاظ سرعت تحلیل برنامه‌ها ۲۷درصد سریع‌تر از ابزار PIAnalyzer عمل می‌کند و می‌تواند آسیب‌پذیری‌های مرتبط با PI را با همان دقت تشخیص دهد. در ادامه پژوهش تلاش می‌گردد که دقت روش ارائه‌شده نیز افزایش یابد.

۷- مراجع

[1] S. Rani and K. S. Dhindsa, “Android Malware Detection in Official and Third Party Application Stores,” International Journal of Advanced Networking and Applications, vol. 9, no. 4, pp. 3506-3509, 2018.

[2] M. Deypir, “Estimating Security Risks of Android Apps Using Information Gain,” Electronic and Cyber Defense, vol. 5, no. 1, pp. 73-83, 2017. [Online]. Available: https://ecdj.ihu.ac.ir/article_200138_30b9b8eeec05f9d21c00f3f4a40d6274.pdf.

[3] A. K. Jha, S. Lee, and W. J. Lee, “Modeling and test case generation of inter-component communication in Android,” in 2015 2nd ACM International Conference on Mobile Software Engineering and Systems, IEEE, pp. 113-116. 2015.

[4] A. Sadeghi, R. Jabbarvand, N. Ghorbani, H. Bagheri, and S. Malek, “A temporal permission analysis and enforcement framework for android,” in Proceedings of the 40th International Conference on Software Engineering, pp. 846-857, 2018.

[5] S. Dhavale and B. Lokhande, “Comnoid: information leakage detection using data flow analysis on android devices,” International Journal of Computer Applications, vol. 134, no. 7, pp. 15-20, 2016.

[6] S. Bugiel, L. Davi, A. Dmitrienko, T. Fischer, and A.-R. Sadeghi, “Xmandroid: A new android evolution to mitigate privilege escalation attacks,” Technische Universität Darmstadt, Technical Report TR-2011-04, 2011.

[7] P. Gadient, M. Ghafari, P. Frischknecht, and O. Nierstrasz, “Security code smells in Android ICC,” Empirical software engineering, vol. 24, no. 5, pp. 3046-3076, 2019.

[8] R. Hay, O. Tripp, and M. Pistoia, “Dynamic detection of inter-application communication vulnerabilities in Android,” in Proceedings of the 2015 International Symposium on Software Testing and Analysis, pp. 118-128, 2015.

[9] S. Groß, A. Tiwari, and C. Hammer, “Pianalyzer: A precise approach for pendingintent vulnerability analysis,” in European Symposium on Research in Computer Security, Springer, pp. 41-59, 2018.

[10] B. Rashidi and C. J. Fung, “A Survey of Android Security Threats and Defenses,” J. Wirel. Mob. Networks Ubiquitous Comput. Dependable Appl., vol. 6, no. 3, pp. 3-35, 2015.

[11] J. A. Shaheen, M. A. Asghar, and A. Hussain, “Android OS with its Architecture and Android Application with Dalvik Virtual Machine Review,” International Journal of Multimedia and Ubiquitous Engineering, vol. 12, no. 7, pp. 19-30, 2017.

[12] E. Chin, A. P. Felt, K. Greenwood, and D. Wagner, “Analyzing inter-application communication in Android,” in Proceedings of the 9th international conference on Mobile systems, applications, and services, pp. 239-252, 2011.

[13] P. Bhiwani and C. Parekh, “Different Android Vulnerabilities,” Advances in Computational Sciences and Technology, vol. 10, no. 5, pp. 1449-1455, 2017.

[14] PendingIntent.html, “http://developer.android.com/reference/android/app/,” 2013.

[15] J. Mitra and V.-P. Ranganath, “Ghera: A repository of android app vulnerability benchmarks,” in Proceedings of the 13th International Conference on Predictive Models and Data Analytics in Software Engineering, pp. 43-52, 2017.

[16] L. Li et al., “Iccta: Detecting inter-component privacy leaks in android apps,” in 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, IEEE, vol. 1, pp. 280-291, 2015.

[17] S. Bhandari et al., “Android app collusion threat and mitigation techniques,” arXiv preprint arXiv:1611.10076, 2016.