بهبود امنیت با استفاده از معیارهای استخراج‌شده از مخازن نرم‌افزاری-معیار فعالیت توسعه‌دهنده

جمیله شفیعی[۱]، دانش‌آموخته کارشناسی ارشد، اشکان سامی[۲]، استادیار

۱- دانشکده مهندسی برق و کامپیوتر - دانشگاه شیراز - شیراز - ایران sami@shirazu.ac.ir–
۲- دانشکده مهندسی برق و کامپیوتر - دانشگاه شیراز - شیراز - ایران - shafiei@cse.shirazu.ac.ir

چکیده: ضعف امنیتی و آسیب‌پذیری در لایه‌های مختلف سیستم‌های نرم‌افزاری، منجر به بسیاری حملات امنیتی برعلیه سیستم‌های نرم‌افزاری می‌شود. برای بهبود امنیت مؤلفه‌های مختلف سیستم، باید به شناسایی و رفع آسیب‌پذیری پرداخت. روش‌هایی که برای شناسایی آسیب‌پذیری نرم‌افزار وجود دارند، به دودسته، روش‌های دستی و خودکار تقسیم می‌شوند. روش‌های دستی، روش‌هایی هستند که از پایش چشمی کد، برای شناسایی آسیب‌پذیری، استفاده می‌کنند. با توجه به دشوار و زمان‌بر بودن این روش‌ها و زمان و منابع محدود تیم‌های توسعه‌دهنده، پژوهشگران به دنبال روش‌هایی برای خودکارسازی شناسایی آسیب‌پذیری نرم‌افزار هستند. ازجمله این روش‌ها، استفاده از معیارهای نرم‌افزاری در مدل‌های شناسایی آسیب‌پذیری است که از چرخه حیات توسعه‌ی نرم‌افزار و مخازن نرم‌افزاری استخراج می‌شوند. معیار فعالیت توسعه‌دهنده، معیاری است که به دلیل تیمی بودن توسعه‌ی نرم‌افزار و نقش برجسته فاکتورهای انسانی، در نوشتن کد، بازبینی، نگهداشت و فرآیند توسعه‌ی نرم‌افزار، نقش مهمی در مدل‌های شناسایی آسیب‌پذیری نرم‌افزار دارد. با توجه به این مطلب، در این مقاله، با طرح یازده فرضیه، به کاوش معیار توسعه‌دهنده از مخزن نرم‌افزاری و بررسی و تحلیل تأثیر این معیار بر روش‌های خودکار شناسایی آسیب‌پذیری نرم‌افزار، بر یک نرم‌افزار متن‌باز (یازده نسخه مرورگر وب موزیلا فایرفاکس)، پرداخته شده است. طی بررسی‌های انجام‌شده جهت شناسایی آسیب‌پذیری، تأثیر چشم‌گیر معیار توسعه‌دهنده بر فایل‌های آسیب‌پذیر، ازنظر آماری معنادار است و این معیار می‌تواند به‌خوبی فایل‌های آسیب‌پذیر را شناسایی کند. بنابراین می‌توان از این معیار در مدل‌های شناسایی خودکار آسیب‌پذیری نرم‌افزار، جهت شناسایی قسمت‌هایی از کد نرم‌افزار که دارای آسیب‌پذیری هستند، استفاده کرد.

واژه‌های کلیدی: معیار توسعه‌دهنده، کاوش مخازن نرم‌افزاری، آسیب‌پذیری نرم‌افزار، پیش‌بینی آسیب‌پذیری، ضعف امنیتی، پیش‌بینی خطا.

Security Improvement with Extracted Metrics of Software Repositories- Developer Activity Metric

J. Shafiei, A. Sami

Faculty of Electrical and Computer Engineering, University of Shiraz, Shiraz, Iran

Abstract: Security weaknesses in the different layers of existing software systems will lead to many cyber attacks against software systems. Software vulnerability is caused due to insecure coding and design. To improve the security of software systems, security weaknesses in the system must be identified and resolved. In general, to reduce number of vulnerabilities several automatic and manual methods have been proposed. Manual methods that are used to identify software vulnerability are using code visual monitoring to identify vulnerabilities. These methods are hard and time consuming. Therefore, researchers seek to provide automated methods for identifying security vulnerabilities. To this end, extracted metrics from software repositories can be used in vulnerability detection models. One of these metrics is Developer Activity Metric. In general, software development is a team work and human factors have a salient role in development process of the software. So these observations naturally lead us to investigate the effect of developer metrics on automatic vulnerability detection methods. In this context, eleven hypotheses on developer metrics have been proposed and statistical analyses on Developer Metrics for eleven versions of Mozilla Firefox are performed. During analysis, we found patterns of developer activity that has statistically significant effect on software vulnerability.

Keywords: Developer activity metrics, mining software repository (MSR), software vulnerability, vulnerability prediction, security weakness, fault prediction.

تاریخ ارسال مقاله: ۹۲/۱۲/۱۵
تاریخ اصلاح مقاله: ۹۳/۰۳/۱۷
تاریخ پذیرش مقاله: ۹۳/۰۶/۲۵
نام نویسنده مسئول: اشکان سامی
نشانی نویسنده مسئول: ایران - شیراز - خیابان ملاصدرا - دانشگاه شیراز - دانشکده فنی کامپیوتر

۱- مقدمه

امروزه، خسارات وارده به سیستم‌های نرم‌افزاری با امنیت پایین و آسیب‌پذیری بالا، به موضوعی مهم تبدیل شده است. تشخیص آسیب‌پذیری نرم‌افزار پیش از انتشار، می‌تواند به میزان زیادی این خسارات را کاهش داده و منجر به بهبود امنیت مؤلفه‌های سیستم شود. به‌طورکلی، یک تعریف منحصربه‌فرد از آسیب‌پذیری نرم‌افزار وجود ندارد. در برخی مطالعات [۱، ۲ و ۳] آسیب‌پذیری را به‌عنوان یک ضعف در نرم‌افزار تعریف می‌کنند که در صورت بهره‌برداری، منجر به خرابی نرم‌افزار می‌شود. با توجه به نبود یک تعریف واحد، اندازه‌گیری هدفمند آسیب‌پذیری یا تعمیم آن دشوار است [۴]. ازجمله بهره‌برداری‌های انجام‌شده از آسیب‌پذیری‌های موجود در نرم‌افزار، می‌توان به موارد زیر اشاره کرد:

در سال ۲۰۰۰ حمله به تجهیزات رادیویی سیستم فاضلاب کوئینزلند استرالیا رخ داد [۵]. طی این حمله حدود ۸۰۰۰۰۰ لیتر فاضلاب خام وارد پارک‌های محلی، رودخانه‌ها و حتی محوطه یک هتل گردید. در سال ۲۰۰۹ جاسوس‌های سایبری به شبکه الکتریکی آمریکا نفوذ کرده [۶] و برنامه‌های نرم‌افزاری به‌جا گذاشتند که می‌توانست سیستم را مختل کند. در سال ۲۰۱۰ پیچیده‌ترین کرم کامپیوتری تحت عنوان استاکس‌نت[۱] شناسایی شد [۷] که هدف آن آسیب زدن به سیستم‌های نرم‌افزاری زیمنس بود. تقریباً ۶۰٪ سیستم‌های آلوده‌شده، در ایران قرار داشتند. در سال ۲۰۱۱ حمله جدیدی تحت عنوان دوکو[۲] شناسایی شد [۸] که ساختار و کد آن شبیه به استاکس‌نت بود و هدف آن ضربه زدن به سیستم هسته‌ای ایران بود. فلیم[۳] هم بدافزار کامپیوتری جدیدی بود که در سال ۲۰۱۲ شناسایی شد [۹] و مانند دوکو هدف جاسوسی داشت. فلیم ازنظر اندازه و پیچیدگی ۲۰ برابر پیچیده‌تر از استاکس‌نت گزارش شده و هدف آن، صدمه زدن به صنایع نفتی ایران بود. با توجه به خسارات وارده، درنتیجه‌ی بهره‌برداری از آسیب‌پذیری‌های موجود در نرم‌افزار، شناسایی و رفع آن، موردتوجه پژوهشگران بسیاری است. روش‌های شناسایی آسیب‌پذیری نرم‌افزار موجود، به دودسته روش‌های دستی و خودکار تقسیم می‌شوند. در روش‌های دستی، پایش کد و شناسایی آسیب‌پذیری، به شکل چشمی انجام می‌گیرد. با توجه به دشوار بودن و صرف هزینه‌های زیاد در این روش، پژوهشگران زیادی در پی خودکارسازی شناسایی آسیب‌پذیری نرم‌افزار هستند. با استفاده از این تکنیک‌ها، متخصصان امنیت می‌توانند مکان‌های آسیب‌پذیر کد را یافته و تلاش‌های آزمون و بررسی خود را بر این نواحی متمرکز کنند [۱۰]. برای افزایش مؤثر بودن و کارآمدی این تلاش، عاقلانه است که این تلاش به نواحی هدایت شود که احتمال خرابی آن‌ها بیش‌تر است یا احتمال یافتن آسیب‌پذیری در این نواحی بیش‌تر است. بدین منظور، یک روش، بررسی تأثیر معیارهای نرم‌افزاری که از مخازن نرم‌افزاری استخراج می‌شوند، بر شناسایی مکان‌های آسیب‌پذیر کد است. معیارهای بسیاری بر آسیب‌پذیری نرم‌افزار تأثیر دارند، نظیر پیچیدگی کد[۴]، انسجام[۵]، ارتباط[۶]، تعداد خط کد[۷]، تغییرات کد[۸] و غیره [۱۶-۱۰]. به‌طورکلی نمی‌توان یک نرم‌افزار را فارغ از تیم توسعه‌ای که آن را به وجود آورده است، در نظر گرفت. با توجه به نقش برجسته توسعه‌دهندگان و فاکتورهای انسانی در توسعه، نگهداشت و بازبینی نرم‌افزار، معیار توسعه‌دهنده، یکی از معیارهایی است که نقش مهمی در شناسایی آسیب‌پذیری نرم‌افزار و بهبود آن دارد [۱۰، ۱۷]. علاوه بر این، توسعه نرم‌افزار به‌طورمعمول کاری تیمی است و ناسازگاری و عدم انسجام در تیم، منجر به ایجاد نرم‌افزاری آسیب‌پذیر می‌گردد. بنابراین معیار توسعه‌دهنده از اهمیت زیادی برخوردار است. گرچه مطالعات بسیاری به بررسی معیار توسعه‌دهنده برای شناسایی خطای نرم‌افزاری پرداخته‌اند [۱۶، ۲۳-۱۸]، اما مطالعات محدودی به شناسایی آسیب‌پذیری با استفاده از این معیار پرداخته‌اند [۱۰، ۱۷]. یافته‌های برخی از این پژوهش‌ها حاکی از تأثیر زیاد معیار توسعه‌دهنده بر مدل‌های پیش‌بینی خطاست [۲۲-۱۸]، اما در برخی موارد نیز بهبود چشم‌گیری حاصل نشده است [۲۳]. علاوه بر این، در برخی پژوهش‌ها، توانمندی مدل‌های پیش‌بینی خطای سنتی برای شناسایی آسیب‌پذیری موردبررسی واقع شده است و یافته‌های آن‌ها حاکی از این است که این مدل‌ها می‌توانند فایل‌های آسیب‌پذیر را نسبت به فایل‌های خطادار، بهتر شناسایی کنند [۱۶]. بنابراین در این مقاله، با توجه به تردیدهایی که در توانمندی معیار توسعه‌دهنده برای بهبود کارایی مدل‌های پیش‌بینی خطا وجود دارد، سعی در استفاده از معیارهای موجود در پیش‌بینی خطا برای پیش‌بینی آسیب‌پذیری شده است و علاوه بر این، برخی معیارهای توسعه‌دهنده نیز جهت شناسایی آسیب‌پذیری نرم‌افزار، پیشنهاد شده‌اند. انگیزه پژوهشی اصلی، بررسی و تحلیل میزان تأثیر معیار توسعه‌دهنده استخراجی از مخازن نرم‌افزاری، بر شناسایی خودکار آسیب‌پذیری نرم‌افزار است.

برای رسیدن به این هدف، آزمایش‌های جامعی جهت بررسی تأثیر معیار توسعه‌دهنده بر روش‌های خودکار شناسایی آسیب‌پذیری نرم‌افزار در یازده نسخه از مرورگر وب موزیلا فایرفاکس [۲۴]، انجام شدند. بدین منظور، یازده فرضیه مطرح شدند که به بررسی تعداد توسعه‌دهندگانی که بر هر فایل کار کرده‌اند، مدت‌زمانی که هر توسعه‌دهنده بر هر فایل کار کرده است، فرکانس فعالیت توسعه‌دهنده بر فایل‌ها، فعالیت همزمان یک توسعه‌دهنده بر چندین فایل، فعالیت توسعه‌دهنده در روزهای پایانی هفته، فعالیت توسعه‌دهنده در زمان‌های خاصی از روز، مکان زندگی و تحصیلات دانشگاهی توسعه‌دهنده و افزوده شدن توسعه‌دهنده جدید به هر نسخه، پرداخته‌اند. یافته‌های این پژوهش نشان می‌دهند که شش فرضیه از مجموع یازده فرضیه‌ی مطرح‌شده، ازنظر آماری معنادار بوده و به‌خوبی فایل‌های آسیب‌پذیر را شناسایی می‌کنند. بنابراین می‌توان از این معیارهای توسعه‌دهنده که در قالب فرضیه مطرح شده‌اند، جهت پیش‌بینی مکان‌های آسیب‌پذیر کد و هدایت سازمان و تیم

توسعه‌دهنده برای صرف زمان و منابع محدود خود بر قسمت‌های آسیب‌پذیرتر کد، استفاده کرد.

بقیه این مقاله به شکل زیر سازمان‌دهی شده است. بخش ۲، به شرح پژوهش‌های مرتبط با این مقاله می‌پردازد که شامل پژوهش‌هایی است که به بررسی شناسایی خطا و آسیب‌پذیری با استفاده از معیار توسعه‌دهنده، می‌پردازند، همچنین پژوهش‌های مربوط به خودکارسازی شناسایی آسیب‌پذیری نیز موردبررسی واقع شده‌اند. در بخش ۳، معیارهای توسعه‌دهنده پیشنهادی ارائه شده‌اند. بخش ۴، نحوه‌ی تولید مجموعه داده را شرح می‌دهد. در بخش ۵، فرضیه‌های پیشنهادی ارائه شده‌اند. بخش ۶، به ارزیابی رویکرد پیشنهادی می‌پردازد و درنهایت نتیجه‌گیری و کارهای آینده ارائه شده‌اند.

۲- کارهای مرتبط

به‌طورکلی توسعه‌ی نرم‌افزار، می‌تواند پیچیده باشد. با توجه به پیچیدگی طراحی یا پیچیدگی برنامه، خطاها و خرابی‌هایی در بسیاری از مراحل چرخه حیات نرم‌افزار ایجاد می‌شوند [۲۵]. عواقب ناشی از خرابی‌های سیستم، که تحت عنوان آسیب‌پذیری نرم‌افزار شناخته می‌شوند، سیاست‌های امنیتی را مخدوش می‌کنند که می‌توانند به شکل یک اشتباه در مشخصات، توسعه یا پیکربندی سیستم باشند. آسیب‌پذیری می‌تواند منجر به از دست رفتن اطلاعات و کاهش ارزش یا مفید بودن سیستم گردد [۲۶، ۲۷]. در این بخش، برخی از پژوهش‌های پیشین که مرتبط با این مقاله هستند، در مورد شناسایی خطا و آسیب‌پذیری با استفاده از معیار توسعه‌دهنده ارائه شده‌اند و برخی از روش‌های خودکار شناسایی آسیب‌پذیری نرم‌افزار نیز موردبررسی واقع شده‌اند.

۲-۱- شناسایی خطا با استفاده از معیار توسعه‌دهنده

پژوهش‌های مختلفی جهت شناسایی خطای نرم‌افزاری با استفاده از معیار توسعه‌دهنده انجام شده است. ازجمله مطالعاتی که بر استفاده از معیار توسعه‌دهنده در شناسایی خطاهای نرم‌افزاری انجام شده‌اند، به شرح زیر می‌باشند.

ناگاپان و همکارانش در سال ۲۰۰۶ [۱۸]، از معیار تعداد توسعه‌دهنده در مجموعه‌ی معیار فرآیند، در ساخت مدل بر اساس داده‌های ویندوز XP، برای پیش‌بینی خرابی‌های ویندوز سرور ۲۰۰۳، استفاده کردند. در این مطالعه از دودسته معیار فرآیند[۹] و معیار محصول[۱۰] استفاده شده است که معیار توسعه‌دهنده تحت معیار فرآیند ارائه شده است. یافته‌های آن‌ها حاکی از شناسایی موفق خرابی‌های بعد از انتشار ویندوز سرور ۲۰۰۳ است. ویوکر و همکارانش در سال ۲۰۰۷ [۱۹]، طی بررسی‌های مختلف برای پیش‌بینی فایل‌هایی که بیش‌ترین تعداد خطا را در انتشار بعدی دارند، تأثیر افزودن معیار فعالیت توسعه‌دهنده را به مدل مدنظر قرار داده‌اند. یافته‌های این پژوهش مؤثر بودن این معیار در پیش‌بینی خطا و افزایش کارایی مدل را نشان می‌دهد. آبرو و همکارانش در سال ۲۰۰۹ [۲۰]، به بررسی فرکانس ارتباطات توسعه‌دهنده و تعداد باگ‌های تزریق‌شده در نرم‌افزار پرداختند. یافته‌های آن‌ها حاکی از وجود همبستگی معناداری میان فرکانس ارتباطات توسعه‌دهنده و تعداد خطاهای تزریق‌شده در نرم‌افزار است.

متسوموتو و همکارانش در سال ۲۰۱۰ [۲۱]، از معیارهایی نظیر تعداد تغییرات کد ایجادشده توسط هر توسعه‌دهنده، تعداد کامیت‌های انجام‌شده توسط هر توسعه‌دهنده و تعداد توسعه‌دهندگان هر ماژول، برای تحلیل رابطه میان تعداد خطاها و معیار توسعه‌دهنده بر مجموعه‌ی داده‌ی پروژه‌ی ایکلیپس [۲۸]، استفاده کردند. یافته‌های این پژوهش، نشان می‌دهد که ماژول‌هایی که توسط توسعه‌دهندگان بیش‌تری تغییر یافته‌اند، خطاهای بیش‌تری دارند و در مقایسه با مدل‌های متداول پیش‌بینی خطا، معیار توسعه‌دهنده، کارایی پیش‌بینی را بهبود داده است. یافته‌های پژوهشی بل و همکارانش در سال ۲۰۱۱ [۲۲]، حاکی از مفید بودن معیار تعداد تجمعی توسعه‌دهندگان در پیش‌بینی خطا است و بیش از ۲٪ بهبود در شناسایی خطا در ۲۰٪ فایل‌هایی که خطادار شناخته شده‌اند، حاصل شده است. اما مطالعه‌ی آن‌ها نشان داد که افزودن اطلاعات در مورد این‌که هر توسعه‌دهنده منحصربه‌فرد کدام فایل را تغییر داده است، پیش‌بینی خطا را بهبود نمی‌دهد. شین و ویلیامز در سال ۲۰۱۱ [۱۶]، به بررسی امکان استفاده از مدل‌های متداول پیش‌بینی خطا، در پیش‌بینی آسیب‌پذیری پرداختند. هدف آن‌ها استفاده از معیارهای پیچیدگی کد، تغییرات کد و تاریخچه‌ی خطا که در مدل‌های سنتی پیش‌بینی خطا وجود دارند، در پیش‌بینی آسیب‌پذیری نرم‌افزار بود. یافته‌های این پژوهش نشان می‌دهد که هر دو مدل پیش‌بینی خطا و پیش‌بینی آسیب‌پذیری، کارایی یکسانی در پیش‌بینی آسیب‌پذیری ارائه می‌کنند. آن‌ها طی آزمایشی دیگری به این نتیجه رسیدند که این معیارها، فایل‌های آسیب‌پذیر را بهتر از فایل‌های خطادار، پیش‌بینی می‌کنند. بنابراین می‌توان از مدل‌های پیش‌بینی خطا برای جایگزینی مدل‌های متداول پیش‌بینی آسیب‌پذیری، استفاده نمود. بنابراین در این مقاله، با توجه به یافته‌های پژوهش [۱۶]، به بررسی تأثیر معیار توسعه‌دهنده بر مدل‌های خودکار شناسایی آسیب‌پذیری نرم‌افزار پرداخته شده است. با توجه به تلاش‌های پژوهشی و صنعتی که برای پیش‌بینی خطای نرم‌افزاری انجام می‌شود، در مطالعه‌ی انجام‌شده توسط جیانگ در سال ۲۰۱۳ [۲۹]، به ساخت مدل پیش‌بینی خطای مجزا برای هر توسعه‌دهنده پرداخته شده است. با توجه به این‌که توسعه‌دهندگان مختلف شیوه‌های کد نویسی مختلف، فرکانس ارتباط و سطح تجربه متفاوتی دارند که منجر به الگوهای خطای متفاوتی می‌شود، این مدل‌های مجزا ساخته شده‌اند. این مدل‌های پیش‌بینی خطا، مدل‌های پیش‌بینی خطای شخصی[۱۱] نامیده می‌شوند. با ترکیب این مدل‌های مختلف، مدلی ایجاد می‌شود که مدل پیش‌بینی خطای ترکیبی مبتنی بر اطمینان[۱۲] نامیده می‌شود. این دو مدل برای کلاس‌بندی خطاها در

سطح فایل، بر شش پروژه‌ی شناخته‌شده استفاده شدند-کرنل لینوکس، ایکلیپس، PostgresSQL، Xorg، Lucene و Jackrabbit. نتایج ارائه‌شده در این مطالعه نشان می‌دهند که مدل ساخته‌شده بهتر از مدل‌های متداول عمل می‌کند. مک اینتاش و همکارانش در سال ۲۰۱۴ [۳۰]، به بررسی رابطه میان کیفیت نرم‌افزار و نسبت تغییراتی که در بازبینی کد ایجاد شده است و مشارکت توسعه‌دهندگان در فرآیند بازبینی کد پرداختند. نتایج آن‌ها نشان می‌دهد که پوشش پایین بازبینی کد و مشارکت توسعه‌دهنده، موجب تولید مؤلفه‌هایی با حداکثر دو تا پنج خطای بعد از انتشار اضافی می‌شوند و کدی که بازبینی آن به شکل ضعیف انجام شده است، تأثیر منفی بر کیفیت نرم‌افزار داشته و منجر به افزایش خطا می‌شود. مطالعه‌ی آن‌ها بر پروژه‌های متن‌باز Qt [۳۱]، VTK [۳۲] و ITK [۳۳] انجام شده است.

۲-۲- شناسایی آسیب‌پذیری با استفاده از معیار توسعه‌دهنده

پژوهش‌های محدودی، توانایی معیار فعالیت توسعه‌دهنده را در پیش‌بینی نقاط ضعف امنیتی نرم‌افزار موردبررسی قرار داده‌اند. پژوهش‌هایی که به بررسی معیار فعالیت توسعه‌دهنده در شناسایی آسیب‌پذیری نرم‌افزار، پرداخته‌اند، به شرح زیر می‌باشند:

منیلی و ویلیامز در سال ۲۰۰۹ [۱۷]، به بررسی نقش جوامعی بزرگ از توسعه‌دهندگان، برای یافتن آسیب‌پذیری در کد پرداختند. یافته‌های آن‌ها نشان می‌دهد درصورتی‌که افراد زیادی کد را پایش کنند، باگ‌های زیادی ایجاد می‌شوند و این مطلب را تحت عنوان قانون لینوس[۱۳]، بیان کردند. در این مطالعه، امنیت سیستم نرم‌افزاری متن‌باز کرنل ۴ لینوکس ردهت، با استخراج معیار فعالیت توسعه‌دهنده، موردبررسی قرار گرفته است و نشان داده شده است، فایل‌هایی که توسط گروه‌های توسعه‌دهنده‌ی مستقل، توسعه داده شده‌اند، از قانون لینوس، پیروی می‌کنند و احتمال وجود آسیب‌پذیری در آن‌ها بیش‌تر است. فایل‌هایی که تغییراتشان توسط ۹ یا تعداد بیش‌تری توسعه‌دهنده اعمال شده است، به میزان ۱۶ برابر، آسیب‌پذیرتر از فایل‌هایی هستند که توسط تعداد کم‌تری از توسعه‌دهندگان، تغییر داده شده‌اند. بنابراین ایجاد تغییرات توسط تعداد زیادی توسعه‌دهنده، بر آسیب‌پذیری سیستم تأثیر مستقیم دارد.

شین و همکارانش در سال ۲۰۱۱ [۱۰]، از معیار توسعه‌دهنده به همراه معیارهای پیچیدگی کد و تغییرات کد در پیش‌بینی فایل‌های آسیب‌پذیر استفاده کردند. آن‌ها آزمایش‌هایشان را بر روی دو پروژه متن‌باز موزیلا فایرفاکس و کرنل لینوکس انجام دادند. اطلاعات مربوط به نقاط ضعف امنیتی این دو پروژه در پایگاه داده مربوط به نقاط ضعف امنیتی موجود است. آن‌ها سه مدل پیش‌بینی کننده با استفاده از هر دسته از معیارهای اندازه‌گیری به‌صورت جداگانه و یک مدل هم با استفاده از ترکیب این معیارها تولید کردند. در این مطالعه، جهت مدل‌سازی از طبقه بندهای شبکه بیزین و آنالیز تفکیک‌کننده[۱۴] استفاده شده است. نتایج به‌دست‌آمده نشان داد که قدرت پیش‌بینی کنندگی معیارها در بین پروژه‌های مختلف و با استفاده از تکنیک‌های مدل‌سازی مختلف، متفاوت است. آن‌ها در بهترین حالت حدود ۷۰٪ از فایل‌های آسیب‌پذیر را با دقت کم‌تر از ۵٪ شناسایی کردند. بوسو و همکارانش در سال ۲۰۱۴، [۳۴]، به تحلیل داده‌ی بازبینی دوتایی[۱۵] کد پروژه‌ی متن‌باز اندروید پرداختند (AOSP). هدف آن‌ها بررسی تولید آسیب‌پذیری امنیتی در اثر تغییرات کد ایجادشده توسط توسعه‌دهندگان، بود. معیار فعالیت توسعه‌دهنده‌ای که موردبررسی قرار گرفت، معیار تغییرات کد ایجادشده توسط توسعه‌دهنده بود. با استفاده از فرآیند تحلیل دستی سیستماتیک، به شناسایی ۶۰ تغییر آسیب‌پذیر کد (VCC[۱۶]) دست یافتند. یافته‌های آن‌ها نشان داد، احتمال ایجاد تغییرات کد آسیب‌پذیر توسط توسعه‌دهندگان پروژه‌ی متن‌باز اندروید، پیش از انتشار AOSP، بیشتر است و بعد از انتشار آن، کاهش می‌یابد.

۲-۳- روش‌های خودکار شناسایی آسیب‌پذیری نرم‌افزار

ججیک و همکارانش در سال ۲۰۰۸ [۱۱]، تعداد هشدارهای یک ابزار تحلیل ایستا، تغییرات کد و تعداد خط کد را به‌عنوان معیارهای اندازه‌گیری ویژگی نرم‌افزار جهت پیش‌بینی نقاط آسیب‌پذیر کد استفاده کردند. آن‌ها با انجام آزمایش روی یک نرم‌افزار مخابراتی تجاری، توانستند مؤلفه‌های آسیب‌پذیر را با نرخ مثبت کاذب[۱۷]، ۸٪ و نرخ منفی کاذب[۱۸]، صفر درصد شناسایی کنند. اطلاعات مربوط به نقاط ضعف امنیتی نرم‌افزار جهت ساخت مجموعه داده موردنیاز برای مدل‌سازی، از بررسی شکست‌های[۱۹] قبل و بعد از انتشار نرم‌افزار به دست آمد.

شین و ویلیامز در سال ۲۰۰۸ [۱۲] ارتباط بین معیارهای اندازه‌گیری پیچیدگی کد و نقاط ضعف امنیتی نرم‌افزار را ارزیابی کردند. آزمایش‌های آن‌ها روی موتور جاوا اسکریپت موزیلا که اطلاعات مربوط به نقاط ضعف امنیتی آن در توصیه‌های امنیتی پایه موزیلا وجود دارد، انجام شد. آزمون وابستگی بین نه معیار اندازه‌گیری پیچیدگی نرم‌افزار و تعداد نقاط ضعف امنیتی نرم‌افزار نشان داد که ارتباط ضعیفی بین این معیارها و مشکلات امنیتی نرم‌افزار وجود دارد.

چودهاری و زولکرنین در سال ۲۰۱۱ [۱۴] معیارهای اندازه‌گیری پیچیدگی، ارتباطات و انسجام نرم‌افزار را جهت پیش‌بینی نقاط ضعف امنیتی نرم‌افزار مورداستفاده قرار دادند. تحقیق آن‌ها بر روی ۵۴ نسخه‌ی موزیلا فایرفاکس انجام شد. آن‌ها جهت ساخت مدل‌های پیش‌بینی کننده از چهار طبقه‌بند C4.5، جنگل تصادفی، رگرسیون منطقی[۲۰] و بیزین ساده، استفاده کردند. سه معیار اندازه‌گیری استفاده‌شده در این تحقیق توانستند تقریباً ۷۵٪ از فایل‌های آسیب‌پذیر را با نرخ تشخیص اشتباه کم‌تر از ۳۰٪ و درستی[۲۱] حدود ۷۴٪ شناسایی کنند. هاوسپیان و همکارانش در سال ۲۰۱۲ [۳۵] روشی جدید جهت شناسایی نقاط ضعف امنیتی نرم‌افزار با استفاده از متن‌کاوی ارائه کردند. آن‌ها کد منبع نرم‌افزار را به‌عنوان یک متن خام

در نظر گرفتند و هر کلمه در متن، به‌عنوان خصیصه در نظر گرفته شد. بر این اساس، هر فایل، به‌صورت برداری از خصیصه‌ها در نظر گرفته شد که مقدار خصیصه‌ها تعداد تکرار هر کلمه در فایل مورد نظر بود. آزمایش‌های آن‌ها بر روی ۱۸ نسخه از نرم‌افزار k9 که یک نرم‌افزار موبایل است، انجام شد و اطلاعات مربوط به نقاط ضعف امنیتی، توسط ابزار Fortify استخراج گردید. آن‌ها توانستند ۸۸٪ از فایل‌های آسیب‌پذیر را با دقت[۲۲] ۸۵٪ شناسایی کنند.

شوای و همکارانش در سال ۲۰۱۳ [۳۶]، یک روش جدید بر اساس الگوریتم ژنتیک و تحلیل پویای ردیابی مقادیر حاصل از ورودی[۲۳] کاربر پیشنهاد کردند. در این مطالعه، در ابتدا تحلیل‌های ایستا برای محاسبه اطلاعات مسیر بحرانی انجام شده‌اند که شامل توابع خطر[۲۴]، توابع دارای تعداد پیچیدگی Cyclomatic بالا و ساختارهای حلقه بودند. دوم، تحلیل ردیابی مقادیر حاصل از ورودی کاربر، برای شناسایی بایت‌های کلیدی برای کاهش فضای ورودی معرفی شدند و سوم تابع برازندگی الگوریتم ژنتیک بر اساس اطلاعات مسیر بحرانی ارائه‌شده و عملگرهای ژنتیک بر فضای ورودی کاهش‌یافته اجرا شده‌اند. نتایج نشان می‌دهند که روش پیشنهادی، دقت تشخیص آسیب‌پذیری بالاتری نسبت به روش‌های معمول به دست می‌آورد.

وو و همکارانش در سال ۲۰۱۴ [۳۷]، به دلیل بالا بودن نرخ مثبت کاذب روش‌های تشخیص آسیب‌پذیری نرم‌افزار، تابع تطبیقی k را پیشنهاد کردند و SVAAKSC را تحت عنوان روش تحلیل آسیب‌پذیری بر اساس خوشه‌بندی دنباله تطبیقی-k ارائه کردند. همه‌ی اشیای موجود در پایگاه داده دنباله آسیب‌پذیری نرم‌افزار SVSD، به k خوشه تقسیم شده‌اند. بعدازآن با تطبیق شباهت‌های میان آسیب‌پذیری تشخیص داده‌شده‌ی نرم‌افزار و هر مرکز خوشه‌بندی، قضاوت در مورد این‌که آسیب‌پذیری تشخیص داده‌شده واقعاً آسیب‌پذیری است یا خیر، انجام شد. نتایجی که روش SVAAKSC پیشنهادی به دست می‌آورد، نرخ کاذب مثبت پایین‌تر و زمان تحلیل بهتری دارد.

۳- معیارهای توسعه‌دهنده پیشنهادی

آسیب‌پذیری نرم‌افزار نه‌تنها به ویژگی‌های خود محصول بستگی دارد، بلکه به ویژگی‌های توسعه‌دهندگانی که محصول را ایجاد کرده‌اند، نیز بستگی دارد [۲۱]. توسعه‌ی نرم‌افزار، به‌طورمعمول، فعالیتی تیمی است و توسط تیم توسعه‌ای انجام می‌شود که باهم بر یک پروژه کار می‌کنند. عدم پیوستگی و انسجام تیم، ارتباطات نادرست و فهم اشتباه، همگی می‌توانند منجر به مشکلات امنیتی شوند. هم‌چنین همکاری ضعیف توسعه‌دهندگان می‌تواند کد نویسی امن را در سراسر پروژه، کاهش دهد. علاوه بر این، عوامل انسانی، بیش‌ترین تراکنش را با کد، چه در مرحله‌ی نوشتن کد و چه در مرحله‌ی بازبینی، پایش، نگهداشت کد و رفع خطا دارند، بنابراین با توجه به اهمیت موارد ذکرشده، در این مطالعه، به بررسی معیار فعالیت توسعه‌دهنده و تأثیر آن بر شناسایی خودکار آسیب‌پذیری نرم‌افزار پرداخته شده است.

ازجمله معیارهای معمول فعالیت توسعه‌دهنده، تعداد تغییرات کد ایجادشده توسط هر توسعه‌دهنده، تعداد نظرات گذاشته‌شده توسط هر توسعه‌دهنده و تعداد توسعه‌دهندگان هر واحد[۲۵] و فایل، هستند [۲۳-۱۶]. با توجه به برجسته بودن نقش این معیار در بررسی‌های انجام‌شده، فرضیه‌هایی برای بررسی آماری این عقیده، به شکل تجربی، ارائه شده‌اند. فرضیه‌های مطرح‌شده در این مقاله، به بررسی معیارهایی نظیر تعداد توسعه‌دهندگانی که بر هر فایل کار کرده‌اند، مدت‌زمانی که هر توسعه‌دهنده بر هر فایل کار کرده است، فرکانس فعالیت توسعه‌دهنده بر فایل‌ها، فعالیت همزمان یک توسعه‌دهنده بر چندین فایل، فعالیت توسعه‌دهنده در روزهای پایانی هفته، فعالیت توسعه‌دهنده در زمان‌های خاصی از روز، مکان زندگی توسعه‌دهنده و تحصیلات دانشگاهی توسعه‌دهندگان، پرداخته‌اند. این اطلاعات از مخزن نرم‌افزاری مرورگر وب موزیلا فایرفاکس استخراج شده است. مخازن نرم‌افزاری، منبعی غنی از اطلاعات را فراهم می‌کنند که می‌توان از این اطلاعات در مواردی نظیر پیش‌بینی خطا، تخمین هزینه و تلاش، پشتیبانی تصمیم‌گیری‌های خودکاری که در سازمان انجام می‌شود و غیره استفاده کرد [۱۶-۱۰، ۳۸]. معیارهای توسعه‌دهنده‌ای که در این مطالعه مورد استفاده واقع شده‌اند، از مخزن CVS[۲۶] مرورگر وب موزیلا فایرفاکس [۳۹] استخراج شده‌اند. اطلاعات توسعه‌دهنده در این مخزن شامل تاریخ و زمانی که هر توسعه‌دهنده بر فایل کار کرده است، نظراتی که ارائه کرده است، تعداد باگ‌ها، نام کاربری توسعه‌دهنده و غیره می‌باشند. با استفاده از این اطلاعات، یازده فرضیه مطرح شده‌اند و با بررسی آماری این فرضیه‌ها به شناسایی مکان‌های آسیب‌پذیر کد پرداخته شده است.

۴- تولید مجموعه داده

برای استخراج و آماده‌سازی مجموعه داده موردنیاز جهت بررسی تأثیر معیار توسعه‌دهنده بر مدل‌های خودکار شناسایی آسیب‌پذیری نرم‌افزار، دو فعالیت مهم باید صورت گیرد: در ابتدا، باید مجموعه کاملی از نقاط ضعف امنیتی استاندارد نرم‌افزار تهیه گردد؛ سپس، اطلاعات مربوط به توسعه‌دهندگانی که بر فایل‌های مختلف، در نسخه‌های مختلف نرم‌افزار کار کرده‌اند، استخراج شود. در ادامه به شرح نحوه استخراج مجموعه‌های داده پرداخته می‌شود.

۴-۱- مجموعه داده: توصیه‌های امنیتی پایه موزیلا و باگزیلا برای مرورگر وب موزیلا فایرفاکس

از پایگاه داده‌ی مرورگر وب موزیلا فایرفاکس به‌عنوان یک پروژه‌ی متن‌باز با مقیاس بزرگ، استفاده شده است. این پروژه به‌خوبی توسط سیستم کنترل نسخه و سیستم ردیابی باگ، مدیریت شده است که امکان جمع‌آوری داده‌ی تجربی قابل‌اعتماد را فراهم می‌کند. در این

مقاله، از یازده نسخه‌ی موزیلا فایرفاکس استفاده شده است (نسخه ۱,۰، نسخه ۱,۰,۳، نسخه ۱,۰,۷، نسخه ۱,۵,۰,۲، نسخه ۱,۵,۰,۵، نسخه ۱,۵,۰,۸، نسخه ۲,۰,۰,۲، نسخه ۲,۰,۰,۵، نسخه ۲,۰,۰,۸، نسخه ۲,۰,۰,۱۱ و نسخه ۲,۰,۰,۱۴). جهت استخراج لیست آسیب‌پذیری‌های استاندارد مرتبط با یازده نسخه مرورگر وب موزیلا فایرفاکس، از آسیب‌پذیری‌های استاندارد گزارش‌شده در توصیه‌های امنیتی پایه موزیلا [۴۰] استفاده می‌شود. جهت استخراج لیست فایل‌های آسیب‌پذیر از آسیب‌پذیری‌های گزارش‌شده، از روش ارائه‌شده در [۱۴] استفاده می‌شود. در این روش، آسیب‌پذیری‌های مربوط به یازده نسخه مدنظر، از سایت توصیه‌های امنیتی پایه موزیلا، استخراج شده و برای هر آسیب‌پذیری، لینک‌های ارتباطی به باگ مرتبط در باگزیلا [۴۱]، که یک سیستم ردیابی خطا است، دنبال می‌شود. سپس، برای هر باگ، فایل‌هایی که در طی رفع هر باگ تغییر یافته‌اند، تعیین شده و شمارنده‌ی آسیب‌پذیری مرتبط با هر فایل، افزایش می‌یابد.

۴-۲- مجموعه داده: مخزن داده برای استخراج اطلاعات توسعه‌دهنده

برای استخراج مجموعه داده‌ی دیگر، متشکل از تعداد توسعه‌دهندگانی که بر هر فایل کار کرده‌اند، از ابزار Bonsai [۳۹] استفاده شده و اطلاعات مربوط به توسعه‌دهندگان از مخزن داده مرورگر وب موزیلا فایرفاکس استخراج می‌شوند. این ابزار توسط مرورگر موزیلا فایرفاکس ارائه شده است و جهت انجام پرس‌وجو روی مخزن داده‌ی موزیلا فایرفاکس به کار می‌رود. اطلاعات موجود در مخزن، در قالب فایل‌های با فرمت اچ‌تی‌ام‌ال هستند، بنابراین با استفاده از کد نویسی، این اطلاعات استخراج شده و جهت پردازش بهتر و تحلیل معیارهای مختلف، در پایگاه داده ذخیره می‌شوند. با انجام این کار، مجموعه داده موردنیاز برای تحلیل معیار توسعه‌دهنده آماده است که شامل نام کاربری هر توسعه‌دهنده، تاریخ و زمان فعالیت وی، شرح فعالیت و تغییراتی که انجام داده است، است.

۴-۳- پیش‌پردازش داده

پیش از استفاده از مجموعه داده‌های حاصل جهت اندازه‌گیری و اجرای پرس‌وجوهای مختلف، طی بررسی‌های چشمی مشخص شد که برخی از توسعه‌دهندگان دارای چندین نام کاربری جهت تراکنش با مخزن نرم‌افزاری هستند. این امر به دلیل فعالیت گسترده‌ی توسعه‌دهندگان از شرکت‌های مختلف و تغییر و تحولاتی که در طی زمان، ازنظر شغلی داشته‌اند، حاصل شده است. به‌عنوان نمونه، در پانزدهم ماه جولای سال ۲۰۰۳ که شرکت موزیلا به‌عنوان موجودیتی مستقل ایجاد شد، بسیاری از توسعه‌دهندگان که به شکل داوطلبانه بر فایل‌های موزیلا فایرفاکس کار می‌کردند، نام کاربری تحت موزیلا دریافت کردند. با انجام این کار، توسعه‌دهندگانی که تا آن زمان با نام‌های کاربری تحت شرکت‌های مختلف با مخزن نرم‌افزاری تراکنش داشتند، اکنون با نام کاربری دیگری به کار بر روی فایل‌ها پرداختند. بدین شکل، در طی پرس‌وجوها، یک توسعه‌دهنده‌ی واحد، ممکن است چندین بار در نظر گرفته شود، چون با نام‌های کاربری مختلفی با مخزن نرم‌افزاری تراکنش داشته است. بنابراین بررسی داده‌ها و معادل‌سازی توسعه‌دهندگان، با جایگزین کردن یک نام کاربری منحصربه‌فرد برای هر توسعه‌دهنده، لازم است. بدین منظور، بررسی‌های لازم انجام گرفته و با استفاده از نام‌های کاربری معادل که در [۴۲] ارائه شده‌اند، معادل‌سازی توسعه‌دهندگان انجام گرفت. انجام این مرحله برای اندازه‌گیری صحیح و دقیق معیار فعالیت توسعه‌دهنده، ضروری است. در ادامه، با استفاده از مجموعه‌های داده حاصل، به بررسی معیار فعالیت توسعه‌دهنده در قالب فرضیه‌های پیشنهادی پرداخته می‌شود.

۵- فرضیه‌ها

نتایج موجود در جدول (۱) ، تأثیر چشمگیر معیار توسعه‌دهنده را بر فایل‌های آسیب‌پذیر نشان می‌دهند. با بررسی نتایج ارائه‌شده در جدول (۱) می‌توان فرضیه‌هایی را جهت بررسی تأثیر معیار فعالیت توسعه‌دهنده بر فایل‌های آسیب‌پذیر ارائه کرد.

اطلاعات موجود در جدول (۱) با اجرای پرس‌وجو بر مجموعه‌های داده که در مرحله‌ی تولید داده حاصل شده‌اند، فراهم گردیده است. برای ارزیابی تأثیر چشم‌گیر این معیار به‌طور تجربی، یازده فرضیه پیشنهاد شده‌اند. این فرضیه‌ها بر اساس میانگین تعداد توسعه‌دهندگان، میانگین مدت‌زمان فعالیت توسعه‌دهنده بر هر فایل، میانگین فرکانس فعالیت توسعه‌دهنده بر هر فایل، فعالیت همزمان یک توسعه‌دهنده بر چندین فایل، فعالیت همزمان چند توسعه‌دهنده بر یک فایل، فعالیت توسعه‌دهنده در روزهای پایانی هفته، فعالیت توسعه‌دهنده در زمان‌های خاصی از روز، مکان زندگی توسعه‌دهنده و تحصیلات دانشگاهی وی و افزوده شدن توسعه‌دهندگان جدید به هر نسخه هستند. همان‌طور که در جدول (۱) می‌توان مشاهده کرد، میانگین تعداد توسعه‌دهندگانی که بر فایل‌های آسیب‌پذیر در یازده نسخه‌ی مرورگر وب موزیلا فایرفاکس کار کرده‌اند، تقریباً سه برابر تعداد توسعه‌دهندگانی است که بر فایل‌های بدون آسیب‌پذیری کار کرده‌اند. بعلاوه، کران بالای متوسط مدت‌زمانی که یک توسعه‌دهنده بر یک فایل آسیب‌پذیر کار کرده است، تقریباً یک و نیم برابر، میزان زمانی است که یک توسعه‌دهنده بر فایل‌های بدون آسیب‌پذیری کار کرده است. هم‌چنین متوسط زمانی که هر توسعه‌دهنده بر فایل‌های آسیب‌پذیر کار کرده است، بر اساس تعداد روزهای دقیق، بیش از چهار برابر، فایل‌های غیرآسیب‌پذیر است. علاوه بر این، میانگین تعداد توسعه‌دهندگان با فعالیت همزمان بر چند فایل نیز در فایل‌های آسیب‌پذیر، تقریباً یک و نیم برابر فایل‌های بدون آسیب‌پذیری است. میانگین فرکانس فعالیت توسعه‌دهنده بر فایل‌های آسیب‌پذیر، تقریباً پنج برابر، فرکانس فعالیت توسعه‌دهنده بر فایل‌های بدون آسیب‌پذیری

است. میانگین فایل‌های با فعالیت همزمان توسعه‌دهندگان نیز در فایل‌های آسیب‌پذیر، حدود دو برابر فایل‌های بدون آسیب‌پذیری است.

حال با توجه به این مشاهدات، فرضیه‌های زیر ارائه می‌شوند:

فرضیه ۱: فایلی که توسط تعداد زیادی توسعه‌دهنده تغییر می‌یابد، احتمال آسیب‌پذیری بیش‌تری دارد.

فرضیه ۲: فایل‌هایی که مدت‌زمان صرف‌شده بر آن‌ها توسط توسعه‌دهنده، بر اساس کران بالا، بیش‌تر است، آسیب‌پذیری بیش‌تری دارند.

فرضیه ۳: فایل‌هایی که مدت‌زمان صرف‌شده بر آن‌ها توسط توسعه‌دهنده، بر اساس تعداد روزهای دقیق، بیش‌تر است، آسیب‌پذیری بیش‌تری دارند.

فرضیه ۴: فایل‌هایی که فرکانس فعالیت توسعه‌دهنده بر آن‌ها بیش‌تر است، آسیب‌پذیرتر هستند.

فرضیه ۵: فایل‌هایی که توسعه‌دهنده به‌طور همزمان بر آن‌ها فعالیت داشته است، آسیب‌پذیرتر هستند.

فرضیه ۶: فعالیت توسعه‌دهنده در روزهای پایانی هفته احتمال آسیب‌پذیری را افزایش می‌دهد.

فرضیه ۷: فعالیت توسعه‌دهنده در ساعات خاصی از روز (به‌طور مثال بعد از ظهرها) میزان آسیب‌پذیری را افزایش می‌دهد.

فرضیه ۸: همزمانی فعالیت چند توسعه‌دهنده بر یک فایل، آسیب‌پذیری را افزایش می‌دهد.

فرضیه ۹: محل سکونت توسعه‌دهندگانی که منجر به آسیب‌پذیری فایل‌ها شده‌اند، کشور خاصی است.

فرضیه ۱۰: توسعه‌دهندگانی که منجر به آسیب‌پذیری شده‌اند، تحصیلات دانشگاهی خاصی داشته‌اند.

فرضیه ۱۱: افزوده شدن توسعه‌دهندگان جدید به هر نسخه، آسیب‌پذیری را افزایش می‌دهد.

جدول (۱): آمار مربوط به معیار توسعه‌دهنده بر یازده نسخه موزیلا فایرفاکس

نسخه / معیار توسعه‌دهنده	۱,۰	۱,۰,۳	۱,۰,۷	۱,۵,۰,۲	۱,۵,۰,۵	۱,۵,۰,۸	۲,۰,۰,۲	۲,۰,۰,۵	۲,۰,۰,۸	۲,۰,۰,۱۱	۲,۰,۰,۱۴
میانگین فرکانس فعالیت توسعه‌دهنده –فایل‌های غیرآسیب‌پذیر	۳۱	۳۲	۳۴	۳۴	۳۵	۳۶	۳۷	۳۹	۳۶	۰	۴۳
میانگین فرکانس فعالیت توسعه‌دهنده-فایل‌های آسیب‌پذیر	۲۶۷	۱۶۶	۱۵۵	۱۷۷	۱۸۲	۲۰۵	۲۰۹	۲۳۳	۱۴۹	۰	۲۳۳
میانگین تعداد فایل‌ها با فعالیت همزمان چند توسعه‌دهنده-فایل‌های غیرآسیب‌پذیر	۴	۵	۵	۵	۶	۶	۶	۶	۵	۰	۵
میانگین تعداد فایل‌ها با فعالیت همزمان چند توسعه‌دهنده-فایل‌های آسیب‌پذیر	۸	۷	۶	۷	۱۶	۱۰	۱۵	۱۶	۱۱	۰	۷
میانگین مدت‌زمان فعالیت هر توسعه‌دهنده بر اساس روزهای دقیق-فایل‌های غیرآسیب‌پذیر	۲۸	۲۸	۲۹	۳۰	۳۱	۳۲	۳۲	۳۴	۲۹	۰	۳۷
میانگین مدت‌زمان فعالیت هر توسعه‌دهنده بر اساس روزهای دقیق-فایل‌های آسیب‌پذیر	۲۱۱	۱۲۷	۱۲۹	۱۴۰	۱۴۶	۱۵۷	۱۶۶	۱۷۷	۱۷۸	۰	۱۹۲
میانگین مدت‌زمان فعالیت هر توسعه‌دهنده بر اساس کران بالا-فایل‌های غیرآسیب‌پذیر	۱۱۸۲	۱۲۰۷	۱۴۳۷	۱۴۴۶	۱۴۹۹	۱۵۶۰	۱۶۵۳	۱۸۰۳	۱۸۹۰	۰	۱۹۱۰
میانگین مدت‌زمان فعالیت هر توسعه‌دهنده بر اساس کران بالا-فایل‌های آسیب‌پذیر	۱۹۲۲	۲۲۷۷	۲۳۶۱	۱۹۹۰	۲۳۴۳	۲۷۰۰	۲۵۲۰	۲۶۵۰	۲۳۴۶	۰	۳۷۰۳
میانگین تعداد توسعه‌دهندگان-فایل‌های غیرآسیب‌پذیر	۹	۸	۹	۹	۱۰	۹	۱۰	۱۰	۸	۰	۱۱
میانگین تعداد توسعه‌دهندگان-فایل‌های آسیب‌پذیر	۴۴	۲۸	۲۹	۲۷	۳۲	۲۲	۳۱	۲۷	۳۰	۰	۲۵

۵-۱- تفسیر فرضیه‌های مطرح‌شده

با توجه به این‌که اگر فایلی توسط تعداد زیادی توسعه‌دهنده تغییر داده شود، کنترل آن و دنبال کردن تغییرات انجام‌شده توسط توسعه‌دهندگان دشوار شده و احتمال آسیب‌پذیری آن بیش‌تر می‌شود، فرضیه یک طرح شده است. علاوه بر این، هر چه تعداد بیش‌تری توسعه‌دهنده روی فایل‌های آسیب‌پذیر کار کرده باشند، بالطبع مدت‌زمان بیش‌تری روی این فایل‌ها نسبت به فایل‌های بدون آسیب‌پذیری صرف شده است، بنابراین بررسی این مدت‌زمان، هم به شکل کران بالا انجام شده است و هم بر اساس تعداد روزهای دقیق که هر توسعه‌دهنده بر هر فایل کار کرده است. محاسبه زمان فعالیت هر توسعه‌دهنده برحسب کران بالا، به شکل کران بالای مدت‌زمان فعالیت هر توسعه‌دهنده در نظر گرفته شده است. ممکن است در بسیاری از روزهایی که در این بازه قرار می‌گیرند، توسعه‌دهنده فعالیتی نداشته باشد اما بازه‌ی بین اولین و آخرین فعالیت وی به‌عنوان این کران بالا در نظر گرفته می‌شود. اما هنگامی‌که مدت‌زمان فعالیت هر توسعه‌دهنده به شکل روزهای دقیق محاسبه می‌شود، فقط تعداد روزهایی در نظر گرفته می‌شوند که توسعه‌دهنده بر فایل‌ها فعالیت داشته است. تفاوت این معیار با معیار فرکانس فعالیت توسعه‌دهنده در این است که ممکن است یک توسعه‌دهنده در یک روز، فعالیت زیادی روی فایل داشته باشد که این امر منجر به تراکنش‌های زیادی با مخزن نرم‌افزاری می‌شود، تعداد این تراکنش‌ها در طی یک روز، تحت عنوان فرکانس فعالیت توسعه‌دهنده در نظر گرفته شده و شمرده می‌شود اما در محاسبه جهت در نظر گرفته شدن در معیار مدت‌زمان فعالیت توسعه‌دهنده، بر اساس روزهای دقیق، فقط یک‌بار در نظر گرفته می‌شوند. بنابراین اگر توسعه‌دهنده‌ای در طی یک روز تراکنش‌های زیادی داشته باشد، فرکانس فعالیت وی بالا می‌رود اما مدت‌زمان فعالیت وی، فقط یک روز در نظر گرفته شده و تعداد دفعات فعالیت بالا بر مدت‌زمان فعالیت، تأثیر نخواهد داشت.

با توجه به این‌که کار همزمان روی تعداد زیادی از فایل‌ها، به دلیل کاهش تمرکز، امکان ایجاد آسیب‌پذیری را افزایش می‌دهد، به بررسی تعداد توسعه‌دهندگانی که به‌طور همزمان روی فایل‌ها کار کرده‌اند، پرداخته شده است. ازجمله فرضیه‌های دیگری که موردبررسی واقع شده است، بررسی فعالیت توسعه‌دهندگانی است که در ساعت خاصی از روز بر فایل‌ها فعالیت داشته‌اند. مثلاً به بررسی میزان فعالیت توسعه‌دهنده در بعدازظهرها پرداخته شده است. هدف از انجام این کار، بررسی احتمال افزایش آسیب‌پذیری، در اثر فعالیت توسعه‌دهنده در ساعات خاصی از روز است. فرضیه دیگر، بررسی همزمانی فعالیت چندین توسعه‌دهنده بر یک فایل است. عملکرد همزمان چندین توسعه‌دهنده بر یک فایل منجر به دشوار شدن کنترل دسترسی به فایل می‌شود و تغییرات همزمانی که توسعه‌دهندگان مختلف در فایل ایجاد می‌کنند، بر یکدیگر تأثیر دارند و این امر پتانسیل آسیب‌پذیری را به دلیل خارج شدن کنترل از دست توسعه‌دهنده و دشوار بودن کنترل تغییرات ایجادشده توسط سایر توسعه‌دهندگان، افزایش می‌دهد. با توجه به مطالب ذکرشده، طرح این فرضیه توجیه‌پذیر است. در فرضیه دیگری به بررسی محل سکونت توسعه‌دهنده پرداخته شده است. هدف از انجام این کار بررسی احتمال افزایش آسیب‌پذیری، تحت تأثیر محل سکونت توسعه‌دهنده است. فرضیه دیگری که ارائه شده، به بررسی تحصیلات دانشگاهی توسعه‌دهندگان مختلف پرداخته است. هدف از ارائه‌ی این فرضیه، بررسی تأثیر تحصیلات دانشگاهی و تجربه‌ی توسعه‌دهندگان در آسیب‌پذیر نمودن فایل‌ها است. فعالیت توسعه‌دهنده در روزهای پایانی هفته، به دلیل کاهش تمرکز در طی این روزها و داشتن مشغله‌های متعدد، نیز ممکن است منجر به افزایش آسیب‌پذیری گردد. این مطلب نیز در قالب یک فرضیه بررسی شده است. فرضیه آخر به بررسی افزوده شدن توسعه‌دهندگان جدید به هر نسخه می‌پردازد. توسعه‌دهندگان جدیدی که به هر نسخه افزوده می‌شوند، به دلیل عدم آشنایی با نحوه کار و تراکنش با فایل‌ها، ممکن است آسیب‌پذیری را افزایش دهند، که این مطلب نیز در قالب یک فرضیه بیان شده است. معیارهایی که در قالب فرضیه ارائه شده‌اند، در جدول (۲) نشان داده شده‌اند.

۵-۲- نرمال‌سازی داده‌ها

با توجه به این‌که در نظر گرفتن داده‌های ارائه‌شده در جدول (۱)، به ازای هر یک از معیارها، ممکن است به‌تنهایی معنی‌دار نباشد، نیاز به نرمال‌سازی نتایج ارائه‌شده با در نظر گرفتن اندازه فایل، است. بدین معنی که ممکن است تعداد توسعه‌دهنده بیش‌تر در فایل‌های آسیب‌پذیر به‌این‌علت باشد که اندازه فایل بزرگ‌تر، نیاز به کار توسعه‌دهنده بیش‌تری بر فایل داشته باشد و این تعداد بیش‌تر توسعه‌دهنده به دلیل اندازه بزرگ‌تر فایل باشد. بنابراین باید اندازه فایل نیز در محاسبات در نظر گرفته شود و تعداد توسعه‌دهندگان با توجه به‌اندازه فایل نرمال‌سازی شود. اندازه فایل‌ها بر اساس تعداد خط کد با استفاده از ابزار Understand 3.0 [۴۳] استخراج شده است. طی بررسی‌های انجام‌شده، اندازه فایل‌های آسیب‌پذیر و بدون آسیب‌پذیری تقریباً به‌طور میانگین باهم برابر است. بنابراین می‌توان به مقایسه این معیارها بدون نگرانی در مورد اندازه فایل پرداخت.

۶- ارزیابی

برای ارزیابی توان تفکیکی معیارهایی که در قالب فرضیه مطرح شدند، به آزمون فرضیه‌های مطرح‌شده، با استفاده از آزمون فرض آماری[۲۷] پرداخته شده است. هدف از انجام این کار، ارزیابی توان هر یک از فرضیه‌های مطرح‌شده در شناسایی و تفکیک فایل‌های آسیب‌پذیر از فایل‌های بدون آسیب‌پذیری است. در آزمون فرض آماری، دو فرض صفر[۲۸] و مقابل[۲۹] مطرح می‌شوند، فرض صفر به آزمون مساوی بودن

جدول (۲): معیار توسعه‌دهنده و فرضیه‌های پیشنهادی

دسته	معیار	فرضیه
معیار توسعه‌دهنده	تعداد توسعه‌دهنده	فایلی که توسط تعداد زیادی توسعه‌دهنده تغییر یافته است، آسیب‌پذیری بیش‌تری دارد.
	مدت‌زمان فعالیت هر توسعه‌دهنده-کران بالا	فایل‌هایی که مدت‌زمان صرف‌شده بر آن‌ها توسط توسعه‌دهنده، بر اساس کران بالا، بیش‌تر است، آسیب‌پذیری بیش‌تری دارند.
	مدت‌زمان فعالیت هر توسعه‌دهنده-روزهای دقیق	فایل‌هایی که مدت‌زمان صرف‌شده بر آن‌ها توسط توسعه‌دهنده، بر اساس تعداد روزهای دقیق، بیش‌تر است، آسیب‌پذیری بیش‌تری دارند.
	فرکانس فعالیت توسعه‌دهنده	فایل‌هایی که فرکانس فعالیت توسعه‌دهنده بر آن‌ها بیش‌تر است، آسیب‌پذیرتر هستند.
	فعالیت همزمان توسعه‌دهنده بر چندین فایل	فعالیت همزمان توسعه‌دهنده بر چندین فایل، آسیب‌پذیری را افزایش می‌دهد.
	فعالیت توسعه‌دهنده در روزهای پایانی هفته	فعالیت توسعه‌دهنده در روزهای پایانی هفته، احتمال آسیب‌پذیری را افزایش می‌دهد.
	فعالیت توسعه‌دهنده در ساعات خاصی از روز	فعالیت توسعه‌دهندگان در ساعات خاصی از روز میزان آسیب‌پذیری را افزایش می‌دهد.
	فعالیت همزمان چند توسعه‌دهنده بر یک فایل	همزمانی فعالیت چندین توسعه‌دهنده بر یک فایل، آسیب‌پذیری را افزایش می‌دهد.
	محل سکونت توسعه‌دهنده	محل سکونت توسعه‌دهندگانی که منجر به آسیب‌پذیری فایل‌ها شده‌اند، کشور خاصی است.
	تحصیلات دانشگاهی توسعه‌دهنده	توسعه‌دهندگانی که منجر به آسیب‌پذیری شده‌اند، تحصیلات دانشگاهی خاصی داشته‌اند.
	افزوده شدن توسعه‌دهندگان جدید به هر نسخه	افزوده شدن توسعه‌دهندگان جدید به هر نسخه، آسیب‌پذیری را افزایش می‌دهد.

بودن میانگین معیارها، در فایل‌های آسیب‌پذیر و بدون آسیب‌پذیری می‌پردازد. در مقابل فرض صفر، فرض مقابل وجود دارد که خلاف این فرض را مدنظر دارد و عدم‌تساوی میانگین معیار توسعه‌دهنده را در فایل‌های آسیب‌پذیر و بدون آسیب‌پذیری، بررسی می‌کند. حال با توجه به توزیع داده‌ها که در شکل (۱) نمایش داده شده است، جهت بررسی معیار فعالیت توسعه‌دهنده، از توزیع نمونه‌گیری Welch's t-test [۴۴] استفاده شده است. این آزمون یک t-test تغییریافته است که میانگین دو نمونه را مقایسه کرده و کارایی خوبی برای توزیع‌های انحرافی[۳۰] با واریانس نابرابر فراهم می‌کند. با استفاده از فرمول Welch، P-مقدار محاسبه می‌شود. P-مقدار حاصل، نشان‌دهنده‌ی رد یا عدم رد فرض صفر است. درصورتی‌که فرض صفر رد شود، فرض مقابل پذیرفته می‌شود. در پژوهش‌ها، این مقدار عموماً ۰/۰۵ در نظر گرفته می‌شود. فرضیه‌های مطرح‌شده، ازنظر توان تشخیصی، زمانی پشتیبانی می‌شوند که نتایج آزمون Welch به‌طور آماری در سطح مقدار $p<0.05$ معنادار باشند. بنابراین درصورتی‌که مقادیر حاصل از محاسبه‌ی P-مقدار از ۰/۰۵ کمتر باشند، به‌خوبی می‌توانند فایل‌های آسیب‌پذیر را از فایل‌های بدون آسیب‌پذیری تفکیک کنند. بنابراین از این معیارها می‌توان در مدل‌های شناسایی آسیب‌پذیری استفاده کرد.

همان‌طور که در شکل (۱) نشان داده شده است، تحلیل اولیه داده‌ها نشان می‌دهد که داده‌ی استفاده‌شده در این مطالعه، واریانس نابرابر داشته و توزیع‌ها دارای انحراف هستند. بنابراین از آزمون Welch's t-test بر فایل‌های آسیب‌پذیر و غیرآسیب‌پذیر استفاده شده است. فرمول (۱) مربوط به آزمون Welch t-test است.

$$t = \frac{\bar{x}_1 - \bar{x}_2}{\sqrt{\frac{S_1^2}{N_1} + \frac{S_2^2}{N_2}}} \quad (۱)$$

که $\bar{X}_i$ ، S_i^2 و N_i، به ترتیب، iامین میانگین نمونه، واریانس نمونه و اندازه نمونه هستند. وقتی‌که t محاسبه شد، این آمارها را می‌توان با توزیع t- برای آزمون فرضیه‌های صفر[۳۱] که میانگین دو جمعیت برابر هستند، استفاده کرد. به‌طورکلی در پژوهش‌ها، زمانی که مقدار P-حاصل، کمتر از ۰/۰۵ باشد، شواهد خوبی برای رد فرضیه صفر و قبول فرض مقابل فراهم شده و یک فرضیه ازنظر توان تفکیکی پشتیبانی می‌شود. از میان فرضیه‌های آزمون شده، فرضیه‌هایی که ازنظر آماری معنادار هستند، در زیر ارائه شده‌اند. همان‌طور که مشاهده می‌شود، شش فرضیه از یازده فرضیه مطرح‌شده، به‌خوبی توانایی شناسایی فایل‌های آسیب‌پذیر از فایل‌های بدون آسیب‌پذیری را دارند.

۶-۱- آیا تعداد زیاد توسعه‌دهندگانی که بر یک فایل کار کرده‌اند، می‌تواند نشانگر مکان‌های آسیب‌پذیر کد باشد؟

همان‌طور که در جدول (۱) نشان داده شده است، میانگین تعداد توسعه‌دهندگانی که بر فایل‌های آسیب‌پذیر کار کرده‌اند، تقریباً سه برابر تعداد توسعه‌دهندگانی است که بر فایل‌های بدون آسیب‌پذیری کار کرده‌اند. برای بررسی اولین فرضیه، آزمون Welch اجرا شد. یافته‌های موجود در جدول (۳) نشان می‌دهند که در همه‌ی نسخه‌های موردبررسی، مقدار P-حاصل کمتر از ۰/۰۵ است و این معیار توانایی تفکیک فایل‌های آسیب‌پذیر را از فایل‌های غیرآسیب‌پذیر در سطح کمتر از ۰/۰۵ دارد. این امر اولین فرضیه را تأئید می‌کند:

فایلی که توسط تعداد توسعه‌دهندگان زیادی تغییر یافته است، آسیب‌پذیری بیش‌تری دارد.

۶-۲- آیا مدت زمانی طولانی که توسعه‌دهنده بر فایل‌های آسیب‌پذیر کار کرده است، معیاری جهت شناسایی فایل‌های آسیب‌پذیر است؟

مدت‌زمانی که یک توسعه‌دهنده بر فایل‌ها کار کرده است، هم بر اساس کران بالا و هم بر اساس روزهای دقیق، موردبررسی قرار گرفته است. نتایج ارائه‌شده در جدول (۱)، نشان می‌دهند که میانگین مدت‌زمانی که یک توسعه‌دهنده، به شکل کران بالا، بر فایل‌های آسیب‌پذیر کار کرده است، تقریباً یک و نیم برابر فایل‌های بدون آسیب‌پذیری است و هم‌چنین میانگین مدت‌زمانی که یک توسعه‌دهنده بر فایل‌های آسیب‌پذیر کار کرده است، بر اساس روزهای دقیق، تقریباً چهار برابر فایل‌های بدون آسیب‌پذیری است. همان‌طور که اشاره شد، برای بررسی توان تفکیکی این معیارها آزمون Welch اجرا شده است. یافته‌های موجود در جدول (۳) نشان می‌دهند که این معیارها می‌توانند به‌خوبی فایل‌های آسیب‌پذیر را از فایل‌های غیرآسیب‌پذیر تشخیص دهند و مقدار-P کمتر از ۰/۰۵ است. بنابراین فرضیه‌های دو و سه تأیید می‌شوند:

مدت‌زمان فعالیت توسعه‌دهنده بر فایل‌های آسیب‌پذیر، به شکل کران بالا و تعداد روزهای دقیق، می‌تواند فایل‌های آسیب‌پذیر را به‌خوبی شناسایی کند.

۶-۳- آیا فرکانس زیاد فعالیت توسعه‌دهنده می‌تواند فایل‌های آسیب‌پذیر را از فایل‌های بدون آسیب‌پذیری شناسایی کند؟

همان‌طور که در جدول (۱) مشاهده می‌شود، میانگین فرکانس فعالیت توسعه‌دهنده در فایل‌های آسیب‌پذیر تقریباً پنج برابر فایل‌های بدون آسیب‌پذیری است، اجرای آزمون Welch نشان می‌دهد که این معیار می‌تواند فایل‌های آسیب‌پذیر را با مقدار-P کم‌تر از ۰/۰۵، به‌خوبی شناسایی کند. بنابراین فرضیه چهار نیز تأیید می‌شود:

معیار فرکانس فعالیت توسعه‌دهنده، می‌تواند به‌خوبی فایل‌های آسیب‌پذیر را از فایل‌های بدون آسیب‌پذیری شناسایی کند.

۶-۴- آیا فعالیت همزمان یک توسعه‌دهنده بر چندین فایل، آسیب‌پذیری را افزایش می‌دهد؟

همان‌طور که در جدول (۱) مشاهده می‌شود، فعالیت همزمان یک توسعه‌دهنده بر چندین فایل، در فایل‌های آسیب‌پذیر تقریباً دو برابر فایل‌های غیرآسیب‌پذیر است. با اجرای آزمون Welch، مقدار-P حاصل، در همه نسخه‌های موردبررسی، کمتر از ۰/۰۵ به دست آمد، بنابراین این معیار می‌تواند به‌خوبی فایل‌های آسیب‌پذیر را شناسایی کند.

فعالیت همزمان یک توسعه‌دهنده بر چندین فایل، آسیب‌پذیری را افزایش می‌دهد.

۶-۵- آیا فعالیت همزمان چند توسعه‌دهنده بر یک فایل آسیب‌پذیری را افزایش می‌دهد؟

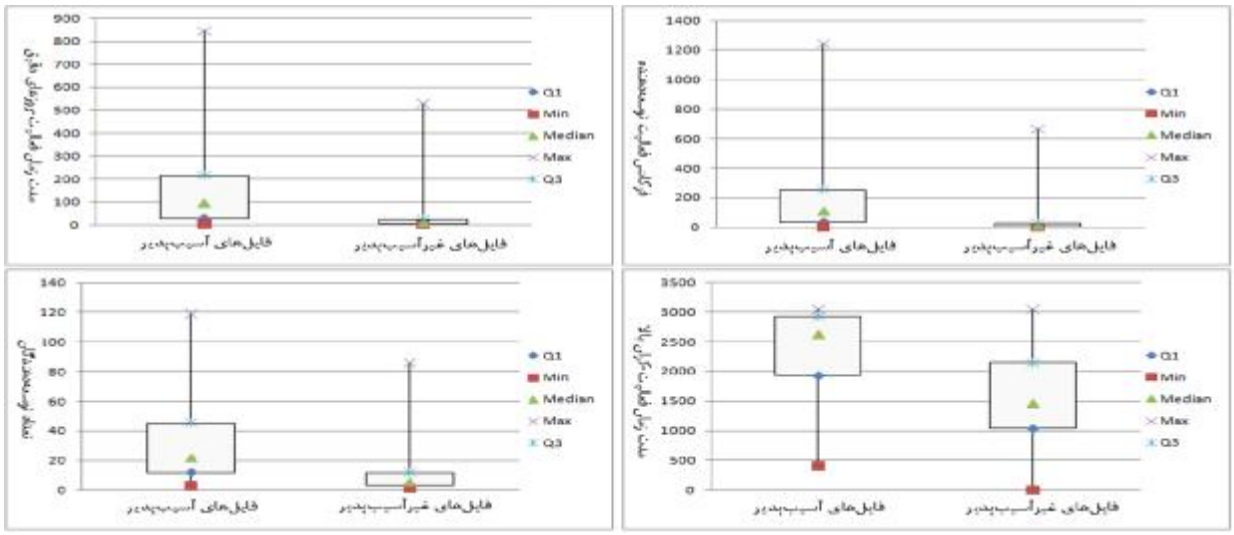
همان‌طور که در جدول (۱) مشاهده می‌شود، میانگین تعداد توسعه‌دهندگان با فعالیت همزمان بر چند فایل، در فایل‌های آسیب‌پذیر تقریباً دو برابر فایل‌های غیرآسیب‌پذیر است. نتایج بررسی تجربی این عقیده با استفاده از آزمون Welch، همان‌طور که در جدول (۳) نشان داده شده است، این فرضیه را تأیید می‌کنند:

فعالیت همزمان چند توسعه‌دهنده بر یک فایل، آسیب‌پذیری را افزایش می‌دهد.

۷- نتیجه‌گیری

شناسایی آسیب‌پذیری نرم‌افزار بعد از انتشار محصول نرم‌افزاری، نه‌تنها مستلزم صرف هزینه‌های زیادی جهت برطرف کردن آسیب‌پذیری است، بلکه منجر به از بین رفتن اعتبار و آبروی شرکت نرم‌افزاری نیز می‌شود. بنابراین پژوهشگران به دنبال روش‌هایی برای شناسایی خودکار آسیب‌پذیری پیش از انتشار نرم‌افزار هستند. یکی از روش‌های موجود، استفاده از معیارهایی است که از چرخه حیات توسعه نرم‌افزار و مخازن نرم‌افزاری استخراج می‌شوند.

در این مقاله، به دلیل اهمیت زیاد عوامل انسانی در مراحل مختلف توسعه نرم‌افزار، نوشتن کد، بازبینی و نگهداشت، به بررسی تأثیر این معیارها در شناسایی آسیب‌پذیری یازده نسخه از مرورگر وب موزیلا فایرفاکس پرداخته شده است. با اجرای آزمون Welch's t-test برای همه‌ی نسخه‌های موردبررسی، مقدار-P کمتر از ۰/۰۵ حاصل شد. بنابراین معیارهای فعالیت توسعه‌دهنده نظیر تعداد توسعه‌دهندگان، مدت‌زمان فعالیت توسعه‌دهنده، تعداد فرکانس فعالیت توسعه‌دهنده، فعالیت همزمان یک توسعه‌دهنده بر چندین فایل و فعالیت همزمان چندین توسعه‌دهنده بر یک فایل، نقش چشم‌گیری در شناسایی مکان‌های آسیب‌پذیر کد دارند و می‌توان از آن‌ها در مدل‌های خودکار شناسایی آسیب‌پذیری استفاده کرد. علاوه بر این، با استفاده از این معیارها می‌توان سازمان و تیم توسعه‌دهنده را در صرف منابع و زمان محدود بر مکان‌های آسیب‌پذیر کد هدایت کرد. به‌عنوان جهت‌های پژوهشی آینده، می‌توان به پیاده‌سازی معیار توسعه‌دهنده، جهت استخراج خودکار معیارها از مجموعه‌های داده مدنظر و استفاده از معیارهای توسعه‌دهنده پیشنهادی همراه با سایر معیارها نظیر پیچیدگی کد، انسجام، ارتباط و غیره، به شکل ترکیبی، در ساخت مدل‌های شناسایی آسیب‌پذیری نرم‌افزار پرداخت.



شکل (۱): توزیع معیار توسعه‌دهنده در فایل‌های آسیب‌پذیر و غیرآسیب‌پذیر

جدول (۳): P-مقدار برای معیار توسعه‌دهنده بر یازده نسخه موزیلا فایرفاکس

نسخه / معیار توسعه‌دهنده	۱	۱,۰,۳	۱,۰,۷	۱,۵,۰,۲	۱,۵,۰,۵	۱,۵,۰,۸	۲,۰,۰,۲	۲,۰,۰,۵	۲,۰,۰,۸	۲,۰,۰,۱۱	۲,۰,۰,۱۴
تعداد توسعه‌دهنده	p<۰/۰۵	p<۰/۰۵	p<۰/۰۵	p<۰/۰۵	p<۰/۰۵	p<۰/۰۵	p<۰/۰۵	p<۰/۰۵	p<۰/۰۵	p<۰/۰۵	p<۰/۰۵
فرکانس فعالیت توسعه‌دهنده	p<۰/۰۵	p<۰/۰۵	p<۰/۰۵	p<۰/۰۵	p<۰/۰۵	p<۰/۰۵	p<۰/۰۵	p<۰/۰۵	p<۰/۰۵	p<۰/۰۵	p<۰/۰۵
فعالیت همزمان چند توسعه‌دهنده بر یک فایل	p<۰/۰۵	p<۰/۰۵	p<۰/۰۵	p<۰/۰۵	p<۰/۰۵	p<۰/۰۵	p<۰/۰۵	p<۰/۰۵	p<۰/۰۵	p<۰/۰۵	p<۰/۰۵
مدت‌زمان فعالیت توسعه‌دهنده- بر اساس روزهای دقیق	p<۰/۰۵	p<۰/۰۵	p<۰/۰۵	p<۰/۰۵	p<۰/۰۵	p<۰/۰۵	p<۰/۰۵	p<۰/۰۵	p<۰/۰۵	p<۰/۰۵	p<۰/۰۵
مدت‌زمان فعالیت توسعه‌دهنده- بر اساس کران بالا	p<۰/۰۵	p<۰/۰۵	p<۰/۰۵	p<۰/۰۵	p<۰/۰۵	p<۰/۰۵	p<۰/۰۵	p<۰/۰۵	p<۰/۰۵	p<۰/۰۵	p<۰/۰۵
فعالیت همزمان یک توسعه‌دهنده بر چند فایل	p<۰/۰۵	p<۰/۰۵	p<۰/۰۵	p<۰/۰۵	p<۰/۰۵	p<۰/۰۵	p<۰/۰۵	p<۰/۰۵	p<۰/۰۵	p<۰/۰۵	p<۰/۰۵

مراجع

[1] S. Kumar and E. Spafford, "A pattern matching model for misuse intrusion detection," 17th National Computer Security Conference, pp. 11-21, 1994.

[2] M. Bishop, "A taxonomy of UNIX system and network vulnerabilities," Tech. Rep. CSE-95-10, Department of Computer Science at the University of California at Davis, 1995.

[3] S. Kumar, *Classification and Detection of Computer Intrusions*, Ph.D. thesis, Purdue University, 1995.

[4] IV. Krsul, *Software Vulnerability Analysis*, PhD thesis, Purdue University, West Lafayette, 1998.

[5] M. Abrams and J. Weiss, "Malicious control system cyber security attack case study – Maroochy Water Service, Australia," http://csrc.nist.gov/groups/SMA/fisma/ics/documents/Maroochy-WaterServices-Case-Study_report.pdf, Accessed March 2014.

[6] S. Gorman, "Electricity grid in U.S. penetrated by spies," http://online.wsj.com/article/SB123914805204099085.html, Accessed March 2014.

[7] N. Falliere, L. O. Murchu and E. Chien, "W32.Stuxnet dossier," Symantec Corp., Security Response, 2011.

[8] Laboratory of Cryptography and System Security (CrySyS), "Duqu: A stuxnet-like malware found in the wild," October 2011.

[9] Flame (Malware), http://en.wikipedia.org/wiki/Flame_%28malware%29#cite_note-16, Accessed March 2014.

[10] Y. Shin, A. Meneely and L. Williams, "Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities," IEEE Transactions on Software Engineering, vol. 37, no. 6, Nov.-Dec 2011.

[11] M. Gegick, L. Williams, J. Osborne and M. Vouk, "Prioritizing software security fortification through code-level metrics," in Proceedings of 4th ACM workshop on Quality of protection, Alexandria, Virginia, USA, pp. 31-38, October 2008.

[12] Y. Shin and L. Williams, "Is complexity really the enemy of software security?," in Proceedings of the 4th ACM Workshop on Quality of Protection, Alexandria, Virginia, USA, pp. 47-50, October 2008.

[13] V. H. Nguyen and L. M. S. Tran, "Predicting vulnerable software components with dependency graphs," in Proceedings of the 6th

International Workshop on Security Measurements and Metrics, Bolzano-Bozen, Italy, September 2010.
[14] I. Chowdhury and M. Zulkernine, "Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities," Journal of Systems Architecture, pp. 294-313, 2011.
[15] Y. Shin and L. Williams, "An initial study on the use of execution complexity metrics as indicators of software vulnerabilities," in Proceedings of the 7th International Workshop on Software Engineering for Secure Systems, Waikiki, Honolulu, Hawaii, pp. 1-7, May 2011.
[16] Y. Shin, and L. Williams, "Can traditional fault prediction models be used for vulnerability prediction?," Empirical Software Engineering, pp. 1-35, 2011.
[17] A. Meneely and L. Williams, "Secure open source collaboration: An empirical study of linus's law," InCCS'09, 2009.
[18] N. Nagappan, T. Ball and B. Murphy, "Using historical in-Process and product metrics for early estimation of software failures," in International Symposium on Software Reliability Engineering, pp. 62-74, 2006.
[19] E. J. Weyuker, T. J. Ostrand and R. M. Bell, "Using developer information as a factor for fault prediction," in Proceedings of the 3rd International Workshop on Predictor Models in Software Engineering, Minneapolis, MN, USA, PROMISE'7, May 2007.
[20] R. Abreu, and R. Premraj, "How developer communication frequency relates to bug introducing changes," in Proceedigs of joint ERCIM Workshop on Software Evolution (EVOL) and Int'l Work-shop on Principles of Software Evolution, pp. 153-157, 2009.
[21] S. Matsumoto, Y. Kamei, A. Monden, K.-I. Matsumoto and M. Nakamura, "An analysis of developer metrics for fault prediction," in Proceedings of the 6th International Conference on Predictive Models in Software Engineering, PROMISE '10, pp. 1, 2010.
[22] R. Bell, T. Ostrand and E. Weyuker, "The limited impact of individual developer data on software defect prediction," Empirical Software Engineering, pp. 1–28, 2011.
[23] E. J. Weyuker, T. J. Ostrand and R. M. Bell, "Do too many cooks spoil the broth? using the number of developers to enhance defect prediction models," Empirical Software Eng., pp. 539-559, 2008.
[24] Mozilla Firefox, www.mozilla.org/en-US, Accessed March 2014.
[25] B. Cashell, W.D. Jackson, M. Jickling and B. Webel, "CRS report for congress: The economic impact of cyber-attacks," Congressional Research Service, April 2004.
[26] S. Conte, H. Dunsmore and V. Shen, *Software Engineering Metrics and Models*, the benjamin/cummings publishing company, 1986.
[27] B. Beizer, *Software Testing Techniques*, electrical engineering/computer science and engineering series. Van nostrand reinhold, 1983.
[28] Eclipse, www.eclipse.org, Accessed March 2014.
[29] T. Jiang, L. Tan, and S. Kim, "Personalized defect prediction," in Proceedings of the 28th International Conference on Automated Software Engineering (ASE'13), Silicon Valley, CA, USA, pp. 279-289, 2013.
[30] S. McIntosh, Y. Kamei, B. Adams, and A. E. Hassan, "The impact of code review coverage and code review participation on software quality, A case study of the Qt, VTK, and ITK projects," MSR '14, May 31 - June 1, 2014, Hyderabad, India.
[31] Qt, http://qt.digia.com, Accessed March 2014.
[32] VTK, http://vtk.org, Accessed March 2014.
[33] ITK, http://itk.org, Accessed March 2014.
[34] A. Bosu, J. C. Carver and M. Hafiz, "When are OSS developers more likely to introduce vulnerable code changes? A case study", OSS 2014, IFIP AICT 427, pp. 234-236, 2014.
[35] A. Hovsepyan, R. Scandariato and W. Joosen, "Software vulnerability prediction using text analysis techniques," IEEE international workshop on security measurements and metrics, Lund, Sweden, pp. 710, September 2012.
[36] B. Shuai, M. Li, H. Li, Q. Zhang and C. Tang, "Software vulnerability detection using genetic algorithm and dynamic taint analysis," Consumer Electronics, Communications and Networks (CECNet), pp. 589-593, November 2013.
[37] D. Wu and J. Ren, "Software vulnerability analysis method based on adaptive-K sequence clustering," TELKOMNIKA Indonesian Journal of Electrical Engineering, vol. 12, no. 6, 2014.
[38] A. E. Hassan, "The road ahead for mining software repositories," in Frontiers of Software Maintenance (FoSM), pp. 48–57, October 2008.
[39] Bonsai, http://www.mozilla.org/projects/bonsai and http://mxr.mozilla.org/mozilla, Accessed March 2014.
[40] Mozilla Foundation Security Advisories (MFSA), http://www.mozilla.org/security/announce, Accessed March 2014.
[41] Bugzilla, https://bugzilla.mozilla.org, Accessed March 2014.
[42] Aliases page of Mozilla Firefox web browser, http://www.ohloh.net/people, Accessed March 2014.
[43] Underestand 3.0, Source code analysis and metrics, http://www.scitools.com, Accessed March 2014.
[44] M.W. Fagerland and L. Sandvik, "Performance of five two-sample location tests for skewed distributions with unequal variances," Contemporary Clinical Trials, Vol. 30, pp. 490–496, 2009.

زیرنویس‌ها

۱ Stuxnet
۲ Duqu
۳ Flame
۴ Code Complexity
۵ Cohesion
۶ Coupling
۷ Line Of Code (LOC)
۸ Code Changes
۹ Process metric
۱۰ Product metric
۱۱ Personalized defect prediction
۱۲ Confidence-based hybrid defect prediction
۱۳ Linus's law
۱۴ Discriminant Analysis
۱۵ Peer review
۱۶ Vulnerable Code Change
۱۷ False positive
۱۸ False negative
۱۹ Failure
۲۰ Logistic Regression
۲۱ Accuracy
۲۲ Precision
۲۳ Taint analysis
۲۴ Danger functions
۲۵ Module
۲۶ Concurrent Versioning System
۲۷ Statistical hypothesis testing
۲۸ Null-hypothesis
۲۹ Alternate-hypothesis
۳۰ Skewed