مجله علمی - پژوهشی "پدافند الکترونیکی و سایبری"
سال چهارم، شماره ۳، پاییز ۱۳۹۵، ص ۶۰-۴۷

ارائه یک رهیافت جدید مبتنی‌بر گراف وابستگی بین فراخوانی‌های سیستمی برای استخراج الگوهای رفتاری مخرب

سعید پارسا[1*]، حسن سیفی[2]، محمدهادی علائیان[3]
۱- دانشیار، ۲- کارشناسی ارشد، ۳- دانشجوی دکتری، دانشگاه علم و صنعت ایران
(دریافت: ۹۴/۱۱/۲۴، پذیرش: ۹۵/۰۵/۱۱)

چکیده

افزایش سریع بدافزارها موجب ناکارآمدی راه‌کارهای شناسایی مبتنی بر امضا و بالعکس مطرح شدن راه‌کارهای کشف مبتنی بر رفتار شده است. درحالی‌که راه‌کارهای کشف مبتنی بر رفتار، راه‌حل‌های امیدوارکننده‌ای در برابر رشد بی‌رویه تولید گونه‌های مختلف از یک خانواده بدافزار هستند اما همچنان از نرخ مثبت کاذب بالایی در کشف بدافزار برخوردار هستند. برای غلبه بر این مشکل امروزه محققان به دنبال شناسایی الگوهایی رفتاری مخربی هستند که به نوعی نشان‌دهنده رفتارهای مخرب ذاتی مربوط به همه نمونه‌های یک خانواده بدافزار باشند. در این مقاله ما یک راه‌کار نوین مبتنی بر کاوش زیر گراف‌های مهم و تعیین زیر گراف‌های تفکیک‌پذیر، برای استخراج دقیق الگوهای رفتاری مخرب موجود در هر خانواده بدافزار، پیشنهاد کرده‌ایم. نتایج حاصل از ارزیابی‌های ما نشان می‌دهد که راه‌کار ما توانسته است الگوهای رفتاری مخرب متمایزکننده موجود در هر خانواده بدافزار، که قابلیت تمیز دادن برنامه‌های مخرب از برنامه‌های سالم را دارند، را با کسب دقت ۹۴٪ در شناسایی بدافزارهای ناشناخته و نرخ خطای کاذب صفر، در مقایسه با نرخ کشف ۵۵٪ در ضد بدافزارهای تجاری و نرخ کشف ۸۶٪ در بهترین کشف‌کننده رفتاری ارائه‌شده تاکنون، استخراج کند.

واژه‌های کلیدی: گراف رفتار، گراف وابستگی بین فراخوانی‌های سیستمی، الگوهای رفتاری مخرب، زیر گراف‌های متمایزکننده، کاوش زیر گراف مهم

۱- مقدمه

رشد بی‌رویه تولید نسخه‌های تغییر شکل یافته بدافزارها، استفاده از راه‌کارهای کشفِ مبتنی‌بر امضا را کاملاً ناکارآمد کرده است. بر طبق نتایج مستند شده شرکت کسپرسکی در سال ۲۰۱۳ نشان می‌دهد که روزانه بیش از ۳۱۵۰۰۰ نمونه بدافزار جدید تولید می‌شود که البته عملکرد و رفتار آن‌ها از پیش وجود داشته است [۱]. در حقیقت، امروزه اکثر بدافزار نویسان از راه‌کارهای گریز پیچیده‌ای (همچون مبهم‌سازی کد[1] [۲]، فشرده‌سازی کد[2] [۳]، چندریختی[3] [۴] و دگرگونی[4] [۵-۴] در سطح کد) برای تولید نسخه‌های جدید از یک بدافزار استفاده می‌کنند. این نسخه‌های جدید، از لحاظ نحوی[5] باهم متفاوت ولی از لحاظ معنایی[6] باهم یکسان هستند و همچنان همان رفتار مخرب اولیه را از خود نشان می‌دهند. به این ترتیب، یک نسخه مبهم‌شده از بدافزار قادر است تا مدت‌ها از دید کشف‌کنندگان مبتنی بر امضا در امان بماند (تا زمانی که یک امضاء جدید برای آن نسخه در پایگاه داده ایجاد شود).

به منظور غلبه بر این مشکل و تأمین حفاظت پیش فعال و بلادرنگ، امروزه محققان به دنبال توسعه راه‌کارهای کشف مبتنی بر رفتار رفته‌اند. این راه‌کارها تشخیص بدافزار را از طریق مقایسه رفتار برنامه‌ها با رفتارهای مخرب از پیش شناخته‌شده انجام می‌دهند. اگرچه راه‌کارهای کشف مبتنی بر رفتار، راه‌حل‌های امیدوارکننده در برابر رشد بی‌رویه تولید گونه‌های مختلف از یک خانواده بدافزار هستند، اما همچنان نرخ مثبت کاذب بالایی در کشف بدافزار دارند [۶].

در این مقاله راه‌کاری نوین به‌منظور استخراج الگوهای رفتاری مخرب موجود در هر خانواده بدافزار ارائه شده است، به‌گونه‌ای هر یک از این الگوها بایستی به‌اندازه کافی کلی انتخاب شود تا بتواند نسخه‌های مبهم‌شده مربوط به آن رفتار مخرب را نیز شناسایی کند و هم این که به اندازه کافی ویژه انتخاب شود تا منجر به ایجاد نرخ خطای مثبت کاذب نشوند. روش ما، مبتنی

* رایانامه نویسنده مسئول: Parsa@iust.ac.ir

1- Code obfuscation
2- Code Packing
3- Polymorphism
4- Metamorphism
5- Syntactically
6- Syntactically

بر کاوش زیرگراف‌های مهم و تعیین زیرگراف‌های است که بیشترین قابلیت تفکیک‌پذیری را در تمیز دادن بدافزارهای یک خانواده از مجموعه برنامه‌های سالم، بر اساس تابع آزمون پیشنهادی H، دارند. که شامل مراحل اساسی زیر است:

- دسته‌بندی بدافزارها بر اساس عملکرد آن‌ها
- استخراج گراف‌های رفتار از برنامه‌های مخرب موجود در هر خانواده و مجموعه برنامه‌های سالم بر اساس سه نوع وابستگی بین فراخوانی‌های سیستمی
- استفاده از یک الگوریتم آماری مبتنی بر کاوش بردار ویژگی، به‌منظور کاوش زیر گراف‌های مهم
- انتخاب K زیر گراف به‌عنوان الگوهای مخرب، که بیشترین قابلیت تفکیک‌پذیری را بر اساس تابع آزمون پیشنهادی H دارند.

نتایج حاصل از ارزیابی‌ها نشان داده است که روش پیشنهادی ما توانسته است تمام ویژگی‌های ذاتی مخرب موجود در هر خانواده بدافزار، که برخی از آن‌ها توسط ضد-بدافزارهای امنیتی نیز گزارش شده‌اند، را به‌درستی شناسایی کند.

در ادامه و در قسمت دوم، چندین راه‌کار مهم و قابلِ قبول در زمینه کشف بدافزار مبتنی بر رفتار برنامه را مورد بررسی قرار داده‌ایم. در قسمت سوم، چارچوب کلی روش پیشنهادی و سپس روال کار را به‌صورت جزئی تشریح کرده‌ایم. در قسمت چهارم، ویژگی‌های محیط پیاده‌سازی و مجموعه داده‌های استفاده شده در ارزیابی را معرفی کرده‌ایم. در قسمت پنجم، به ارزیابی روش مطرح شده در شناسایی الگوهای رفتاری مخرب پرداخته‌ایم. قسمت ششم و هفتم را به نتیجه‌گیری و بیان کارهای آتی اختصاص داده‌ایم.

۲- کارهای مرتبط

به طور کلی امروزه محققان به دنبال ارائه راه حل هایی برای تسخیرِ الگوهای مخربی هستند که خصیصه‌ی ذاتی یک برنامه‌ی مخرب را نشان دهند، و به این ترتیب بتوانند با ایجاد یک رهیافت مؤثر[1] و کارا[2] در مقابل رشد بی‌رویه بدافزارهای مبهم‌شده ایستادگی کنند. مدل‌های غیرموثر، مدل‌هایی هستند که خصوصیات ذاتی یک برنامه مخرب را دریافت نمی‌کنند، بلکه تنها خصوصیاتِ رفتاری مربوط به یک برنامه‌ی مخرب خاص را نشان می‌دهند. در نتیجه، این گونه مدل‌ها نمی‌توانند خیلی در مقابل نسخه‌های مبهم‌شده و چندشکل‌شده از یک بدافزار کارایی خوبی داشته باشند. به‌عنوان مثال، اکثر نرم‌افزارهای ضد-بدافزار سنتی برای تشخیص بدافزار متکی به امضاءهای بایتی (یا دستورالعملی) هستند. در واقع این گونه نرم‌افزارهای ضد-بدافزاری که مدلِ کشف بدافزار خود را بر اساس الگوهای بایتی کد برنامه (امضاء بایتی) انتخاب می‌کنند، به‌راحتی به‌وسیله‌ی راه‌کارهای مبهم‌سازی و چندریختی کد فریب می‌خورند. چرا که راه‌کارهای مبهم‌سازی قادرند به‌آسانی امضای بایتی برنامه‌ها را در حالی تغییر دهند که معنای واقعی برنامه تغییری نکند [۷].

در همین راستا به‌منظور ایجاد یک مُدل رفتاری مؤثر، کشف کنندگانی نیز نمود پیدا کردند [۸-۷] که از تحلیل پیچیده ایستا، برای استخراج معنای یک برنامه، به‌منظور شناسایی بدافزار استفاده می‌کنند. بنابراین از آنجا که تمرکز این راه‌کارها بر معنای واقعی یک برنامه است، در نتیجه برای نمونه بدافزارهای که از راه‌کارهای ساده مبهم‌سازی[3] و چندریختی کد برای تغییر ظاهر خودشان استفاده می‌کنند، خیلی مؤثر هستند [۹]. اما مشکلِ اصلی راه‌کارهای ایستا این است که رسیدن به رفتار واقعی برنامه در سطح ایستا کاملاً دشوار است. این دشواری با رمزنگاری[4] و خود-تغییری[5] در حینِ اجرا، تشدید خواهد شد. علاوه بر این، تحلیلی پر هزینه است، لذا برای جایگزینی با کشف کنندگان ضد-بدافزاری تجاری امروزی که نیازمند پویشِ سریع تعداد زیادی فایل است، مناسب نیست.

به‌طور کلی، روش‌های تحلیل ایستا برای مدل‌سازی رفتار بدافزار مناسب نیستند، و به همین دلیل است که امروزه محققان به ارائه و توسعه‌ی ابزارهای تحلیل پویا، که رفتار برنامه را در حین اجرا زیرِ نظر می‌گیرند، پرداخته‌اند [۹]. بر این اساس، راه‌کارهای دیگری در زمینه‌ی تحلیل پویای بدافزار ایجاد شده‌اند، که بازهم در کشف بدافزار خیلی مؤثر عمل نمی‌کنند. این راه‌کارها، رفتار برنامه را به‌صورت دنباله‌ی از فراخوانی‌های سیستمی[6]، که یک برنامه در زمان اجرا درخواست می‌کند، مُدل می‌کنند. در صورتی که در تشکیل این دنباله وابستگی بین فراخوانی‌های سیستمی لحاظ نشود، یعنی فراخوانی‌های سیستمی به‌صورت مستقل در نظر گرفته شوند، با تغییر ترتیبِ[7] فراخوانی‌های سیستمی یا درجِ فراخوانی‌های سیستمی نامربوط[8] در این دنباله، می‌توان این راه‌کارها را شکست داد [۱۲-۱۰].

کولبیتج و همکاران [۱۳]، به بررسی مشکل حاصل از نمایش خصوصیات رفتاری برنامه بر اساس دنباله اجرایی[9] پرداختند. آن-آن‌ها از یک راه‌کار مشابه با روش پیشنهادشده در این مقاله، برای نمایش رفتار برنامه بر اساس گراف، استفاده کردند. با این تفاوت که در راه‌کار آن‌ها، از زبانِ پیچیده‌تری برای توصیف قیود، بر روی وابستگی‌ها، استفاده شده است. اساسِ کار این رهیافت مبتنی بر

1- Effective
2- Efficient

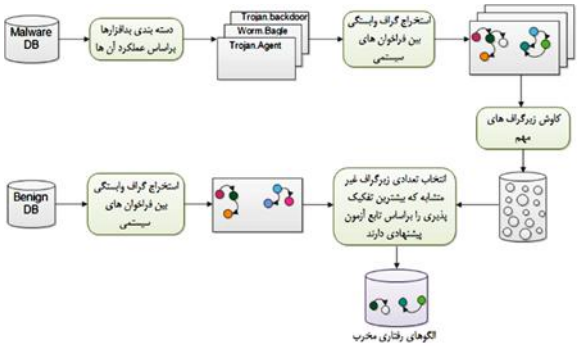
3- Obfuscation
4- Encryption
5- Self-modifiying
6- System Calls Sequence
7- Ordering
8- Junk system-calls insertion
9- Execution Sequences

تعاملاتی است که برنامه با محیط خودش، یعنی سیستم عامل، صورت می‌دهد. این تعاملات از طریق فراخوانی‌های سیستمی صورت می‌گیرد. در این راه‌کار برای شناسایی مجموعه خصوصیات مخرب، برنامه‌های مخرب در یک محیط کنترل شده اجرا می‌شوند. به این ترتیب، از ردیابی جریانِ داده پویا یا لکه‌گذاری برای توصیف دقیق وابستگی‌ها در تشکیل گراف وابستگی بینِ فراخوانی‌های سیستمی استفاده می‌کند. البته برای مطابقت این خصوصیات مخرب با رفتار زمان اجرای برنامه از لکه‌گذاری، به دلیل کند بودن، استفاده نمی‌شود. بلکه تنها فراخوانی‌های سیستمی و پارامترهایشان موردِ تحلیل قرار می‌گیرد، نه این‌که همه اطلاعات را در سطح دستورالعمل، در زمانِ اجرا، تحلیل کند. این ایده منجر به کارا شدنِ عمل شناسایی بدافزار، بدون نیاز به استفاده از عمل گرانِ لکه‌گذاری و استفاده از محیط‌های ویژه (ماشینِ مجازی)، در زمان اجرا واقعی، شده است. در نهایت، نویسنده اذعان دارد که توانسته‌اند عمل کشفِ بدافزار را به‌صورت مؤثر در یک سیستم پایانی با نرخ خطای مثبت کاذب صفر انجام دهند. علاوه بر تفاوتی که این راه‌کار با روش پیشنهادی ما، در نحوه تشکیل گراف رفتار دارد، الگوریتم استخراج خصوصیات مخرب آن نیز در یک راه اساسی با ما تفاوت دارد؛ و آن‌هم این‌که در روش آن‌ها، هیچ‌گونه تلاشی برای تمیز دادن رفتارهای مخرب از مجموعه رفتارهای مشاهده‌شده از برنامه‌های سالم، صورت نمی‌گیرد. بنابراین راه‌کار آن‌ها یقیناً نمی‌تواند تضمین‌کننده نرخ خطای مثبتِ کاذب صفر باشد.

اخیراً راه‌کار جدیدی در [۱۴] توسط Fredrikson و همکاران، مبتنی بر گراف‌کاوی[1] و تحلیل مفهوم[2] برای استخراج خصیصه‌های متمایز مخرب ارائه شده است. این راه‌کار نیز در یک راه کاملاً متفاوتی نسبت به روشِ پیشنهادی ما در این مقاله، عمل استخراج گرافِ رفتار و استخراج خصیصه‌های بهینه مخرب را انجام می‌دهد.

۳- طرح پیشنهادی

فرایند طرح پیشنهادی به‌منظور شناسایی و استخراج الگوهای رفتاری مخرب در شکل (۱) نشان داده شده است. این فرایند، شامل چندین مرحله است. در مرحله اول، بایستی بدافزارها را بر اساس عملکرد آن‌ها دسته‌بندی[3] کرد. هدف از دسته‌بندی بدافزارها، استخراج رفتارهای مخربِ مختص به هر خانواده است. در مرحله دوم، گراف رفتار مربوط به تمام بدافزارها و برنامه‌های سالم که مبتنی بر وابستگی بین فراخوانی‌های سیستمی است، در حین اجرا استخراج می‌شود. در مرحله سوم، از یک الگوریتم آماری مبتنی بر کاوش زیر گراف‌های پرتکرار و معیار اهمیت آماری، برای کاوش زیر گراف‌های مهم[4] استفاده می‌شود. در مرحله چهارم، برای هر خانواده از بدافزارها K زیر گراف مهم غیر متشابه به‌گونه‌ای انتخاب می‌شوند که، بر اساس تابع آزمون H، بیشترین تفکیک‌پذیری را در تمیز دادن برنامه‌های مخرب از برنامه‌های سالم داشته باشند.



شکل (۱). فرایند طرح پیشنهادی به منظور استخراج الگوهای رفتاری مخرب

۳-۱- دسته‌بندی بدافزارها

به‌منظور استخراج رفتارهای مخرب مختص به هر خانواده بدافزار لازم است که ابتدا بدافزارها را بر اساس عملکردشان دسته‌بندی کرد. چرا که در حالت عادی دشوار به نظر می‌رسد که بتوان برای همه خانواده‌های مختلف بدافزار رفتارهای مشترکی را استخراج کرد. به این دلیل که اساساً، هر یک از خانواده‌های مختلف بدافزار، آسیب‌پذیری‌های متفاوتی را مورد هدف قرار می‌دهد و گستره عملکردی مختلفی نسبت به مابقی خانواده‌ها دارد. عمل دسته‌بندی بدافزارها هم می‌تواند به‌صورت دستی صورت گیرد، و هم این که می‌توان از دسته‌بندی بر اساس روش‌های خودکار خوشه‌بندی مبتنی بر رفتار [۱۵-۱۷] و یا دسته‌بندی بر اساس میزان بیشترین شباهت گراف‌ها [۲۰] برای انجام این کار بهره گرفت. ما دسته‌بندی بدافزارها را به‌صورت دستی و بر اساس گزارش‌های ارائه شده توسط ضد- بدافزار کسپرسکی، انجام داده‌ایم؛ و از آن جا که این گزارش‌ها توسط محققان امنیتی و بر اساس مشاهده عملکرد آن بدافزار در حین اجرا و در یک محیط واقعی صورت می‌گیرد، بسیار دقیق‌تر از روش‌های خودکار خوشه‌بندی یا دسته‌بندی است [۱۳-۱۴].

۳-۲- استخراج گراف‌های رفتار

با توجه به این که تمام فعالیت‌های مخربی که یک بدافزار انجام می‌دهد قطعاً در گرو تعامل با سیستم‌عامل است، نمایش

1- Graph Mining
2- Concept Analysis
3- Classification
4- Significant Subgraphs

رفتار یک برنامه بر اساس تعاملاتی که آن برنامه با سیستم‌عامل از طریق فراخوانی‌های سیستمی انجام می‌دهد، توانایی قدرتمندی را در کشف بدافزار مبتنی بر رفتار ایجاد کرده است. در حقیقت می‌توانیم با تشکیل یک گراف از وابستگی‌های بین فراخوانی‌های سیستمی، رفتار برنامه را به‌خوبی مدل کرد. ما برای رهگیری رفتار برنامه‌ها در ضمن اجرا، از راه‌کار قلاب اندازی به جدول SSDT استفاده کرده‌ایم. برای این منظور یک راه‌انداز به نام ProcessHooking.sys در هسته سیستم‌عامل ایجاد شده است، که اقدام به قلاب اندازی به ۴۲ تابع حساس و مهم مورد نظر می‌کند. معیار انتخاب این ۴۲ تابع بر اساس ملاحظات انجام شده بر روی برخی ضد-بدافزارهای رفتاری همچون Kaspersky و BitDefender بوده است. اسامی این ۴۲ تابع به همراه آرگومان‌هایی که برای تشکیل گراف رفتار برنامه مورد استفاده قرار گرفته‌اند، در پیوست آمده است.

- **استخراج وابستگی‌ها:** روش پیشنهادی توانسته است با در نظر گرفتن سه نوع وابستگی بین فراخوانی‌های سیستمی، رفتار برنامه را به‌خوبی به‌صورت گراف مُدل کند. به‌ازای هر آرگومان ابتدا نوع آرگومان و در داخل جفت براکت، ورودی (in) و یا خروجی (out) بودن آن آرگومان مشخص شده است. ۳ نوع وابستگی در نظر گرفته شده برای ایجاد گراف رفتار مبتنی بر وابستگی بین فراخوانی‌های سیستمی عبارت‌اند از:

۱. Handle: تقریباً برای دسترسی به هر شیء و منبعی در سیستم‌عامل (شامل فایل، حافظه، پورت، رجیستری و ...) از handle استفاده می‌شود. این نوع وابستگی با برچسب h در گراف وابستگی نشان داده شده است.

۲. Pvoid: اشاره‌گری است که به ابتدای یک حافظه اشاره دارد. به‌عنوان مثال، فراخوانی سیستمی NtAllocateVirtualMemory یک اشاره‌گر از نوع pvoid را به‌عنوان خروجی (out) برمی‌گرداند که به ابتدای حافظه‌ی تخصیص یافته اشاره دارد، و در ادامه این اشاره‌گر می‌تواند به‌عنوان آرگومان ورودی به تابعی مثل NtProectVirtualMemory ارسال شود. که در واقع چنین رفتاری، نشانگر این است که تدابیر امنیتی خاصی بر روی یک قسمت تخصیص‌یافته از حافظه اعمال می‌شود. این نوع وابستگی با برچسب p در گراف وابستگی نشان داده شده است.

۳. Pclient_id: این آرگومان نیز یک اشاره‌گر به داده ساختاری است که شامل دو فیلد ProcessId و ThreadId است که به ترتیب شناسه پروسه و شناسه نخ مربوطه را نشان می‌دهند. این داده ساختار می‌تواند توسط تابع NtCreateThread تولید شود (out) و توسط NtOpenProcess استفاده شود (In). این نوع وابستگی با برچسب c در گراف وابستگی نشان داده شده است.

- **تعریف رسمی گراف رفتار**

گراف رفتار یک گراف مبتنی بر وابستگی بین فراخوانی‌های سیستمی است که به‌صورت یک ۵ تایی $G = (V, E, L, \alpha, \beta)$ نشان داده می‌شود. در این تعریف،

- V یک مجموعه‌ای از گره‌های مربوط به فراخوانی‌های سیستمی Σ است،
- $E \subseteq V \times V$ یک مجموعه‌ای از یال‌ها است که وابستگی‌های بین فراخوانی‌های سیستمی را نشان می‌دهد،
- L یک مجموعه‌ای از برچسب‌ها است،
- $\alpha : V \cup E \to L$، یک تابع است که برچسب مربوط به یال‌ها و گره‌ها را مشخص می‌کند. به‌گونه‌ای که برچسب هر گره را به نام فراخوانی سیستمی مربوطه‌اش، و برچسب هر یال‌ها را با وابستگی موجود بین دو فراخوانی سیستمی مربوطه‌اش قرار می‌دهد.
- $\beta : E \to \delta_{dep}$ ، یک تابع منطقی است که تمام وابستگی‌ها اعم از ۳ وابستگی اصلی (با برچسب یال h، p و c) به‌علاوه‌ی وابستگی بین دو CC پشتِ سر هم (با برچسب یال $) را به ازای هر یال (v_i, v_j)، در بین دو فراخوانی سیستمی، بیان می‌کند.

با معرفی Σ به‌عنوان مجموعه‌ای تمام فراخوانی‌های سیستمی رهگیری‌شده (۴۲ تابع فهرست شده در پیوست)، می‌توان مجموعه L را به‌صورت $L = \Sigma \cup \{h, p, c, \$\}$ بیان کرد. همان‌طور که پیش از این بیان شد، h، p و c، سه وابستگی اصلی را بین فراخونی‌های سیستمی نشان می‌دهند. برچسب $ هم برای اتصال دو CC پشت سرهم، بین دو فراخوانی سیستمی ابتدایی مربوط به هر CC، استفاده شده است.

یک فراخوانی سیستمی $\sigma \in \Sigma$ یک تابع با n آرگومان ورودی از سه نوع منحصر به فرد (شامل سه نوع handle، pvoid و pclient_id) است، که به‌صورت $\sigma: a_1: t_1, a_2: t_2, \dots, a_n: t_n \to t_r$ تعریف می‌شود. که در آن t_i نوع آرگومان i ام را نشان می‌دهد، و t_r نوع برگشتی تابع را نشان می‌دهد. البته از نوع برگشتی توابع برای استخراج وابستگی استفاده نشده است. از آن جا که نتیجه اجرای یک برنامه P همواره یک دنباله‌ای از درخواست‌ها برای فراخوانی‌های سیستمی به‌صورت $T = (\sigma_1, \sigma_2, \dots, \sigma_n)$ است، می‌توان گراف رفتار برنامه P را به‌صورت $G = (V, E, L, \alpha, \beta)$

بیان کرد به گونه‌ای که:

۱. قرارگرفتنِ هر فراخوانی سیستمی در گرافِ رفتار، منوط به درخواست آن فراخوانی سیستمی در دنباله اجرایی T است به گونه‌ای که:

$$\forall v_i \epsilon V \,.\, \exists \sigma_i \epsilon T \mid \alpha(v_i) = \sigma_i$$

۲. قرارگرفتنِ هر وابستگی در گراف رفتار (به جزء وابستگی $)، منوط به درخواست دو فراخوانی سیستمی در دنباله‌ای اجرایی T به گونه‌ای است که:

$$\forall (v_i, v_j) \epsilon E \,.\, \exists \sigma_i, \sigma_j \epsilon T \mid \alpha(v_i) =$$

$$\sigma_i \wedge \alpha(v_j) = \sigma_j \wedge \beta(v_i, v_j)$$

۳. همان‌طور که گفته شد، وابستگی $ نیز بینِ دو رأسِ ابتدایی دو CC پشت سرهم تشکیل می‌شود.

۳-۳- کاوش زیرگراف‌های مهم

پیش از پرداختن به جزئیات کار، لازم است مفاهیمی را که در ادامه با آن‌ها روبرو می‌شوید، تعریف کنیم.

نرخ فراوانی: با داشتن یک مجموعه گراف G، نرخ فراوانی یک الگوی زیرگراف p برابر است با نسبت تعداد گراف‌های که p را پوشش می‌دهند به تعداد کل گراف‌ها در G.

مجموعه مخرب و مجموعه‌ی سالم: مجموعه‌ی مخرب شامل یک مجموعه گراف است که هر گراف رفتار مشاهده شده از یک برنامه مخرب را نشان می‌دهد. مجموعه سالم نیز شامل یک مجموعه گراف استخراج شده از برنامه‌های سالم است. به همین ترتیب، نرخ فراوانی یک الگوی زیرگراف g در مجموعه مخرب به‌صورت $p(g)$ و در مجموعه سالم به‌صورت $q(g)$ نشان داده می‌شود.

کاوش زیرگراف‌های مهم: با داشتن یک مجموعه دادگان گراف $D = \{G_1, \dots, G_n\}$ و یک تابع هدف F ، این مسئله شامل پیدا کردن همه‌ی زیرگراف‌های g با $F(g) \geq \delta$ است که در آن δ یک آستانه‌ی اهمیت است.

زیرگراف پرتکرار[1]: با داشتن یک مجموعه دادگان گراف $D = \{G_1, \dots, G_2)$ و یک آستانه فراوانی θ، یک زیرگراف g با مقدار پوشش مشاهده‌شده‌ی μ_0 پرتکرار است اگر و فقط اگر $\mu_0 \geq \frac{\theta |D|}{100}$.

اهمیت آماری[2]: اهمیت آماری (یا p-value) یک زیرگراف g با مقدار پوشش مشاهده‌شده μ_0 به صورت احتمال این که این زیرگراف در یک پایگاه داده تصادفی با مقدار پوشش μ رخ می‌دهد، بیان می‌شود که در آن، $\mu \geq \mu_0$ می‌باشد.

امتیاز تفکیک‌سازی[3]: امتیاز تفکیک‌سازی یک الگوی زیرگراف g یک تابع هدف تعریف‌شده توسط کاربر، بر اساس نرخ فراوانی g در مجموعه مخرب $p(g)$ و مجموعه سالم $q(g)$، است. به صورت کلی، تابع هدف به‌گونه‌ای عمل می‌کند که هر چه $p(g)$ از $q(g)$ بیشتر باشد، امتیاز تفکیک‌سازی زیرگراف g بهتر خواهد بود.

در این مرحله، ما به‌ازای هر خانواده بدافزار یک مجموعه زیرگراف مهم استخراج می‌کنیم. این مجموعه زیرگراف مهم شامل زیرگراف‌هایی است که احتمال مشاهده آنها در یک مجموعه گراف تصادفی دیگر، خیلی زیاد نباشد. این شرط توسط اهمیت آماری، که به‌وسیله خود کاربر مشخص می‌شود، کنترل می‌شود.

ما از یک الگوریتم آماری مبتنی بر کاوش بردار ویژگی [۱۹] برای استخراج زیرگراف‌های مهم استفاده کردیم. در این الگوریتم با توجه به مجموعه دادگان، یک سری ویژگی‌های ساختاری ساده شامل گره‌ها و یال‌ها با برچسب‌های خاص از پیش تعریف می‌شوند. سپس هر گره در هر گراف را با یک بردار ویژگی، براساس این ویژگی‌های از پیش تعیین شده، نشان می‌دهند. بردار ویژگی مربوط به هر گره یک ناحیه را در داخل گراف نشان می‌دهد، که چگونگی توزیع ویژگی‌ها در اطراف آن گره را نشان می‌دهد. پس از تعیین ویژگی‌های مربوط به بردار ویژگی (در اینجا شامل همه‌ی فراخوانی‌های سیستمی و همه یال‌های امکان‌پذیر بین فراخوانی‌های سیستمی است)، ابتدا گراف‌ها با قدم‌زنی تصادفی و شروع مجدد[4] بر روی هر گره، به بردارهای ویژگی تبدیل می‌شوند. این مرحله را می‌توان به این شکل توصیف کرد که یک پنجره بر روی هر گره قرار داده می‌شود و سپس یال‌های داخل این پنجره به‌صورت تصادفی پیمایش می‌شوند؛ در ضمن همین پیمایش است که مقداردهی به ویژگی‌های بردار ویژگی مربوط به این گره صورت می‌گیرد. سپس با کاوش بردار ویژگی، بردارهای زیر-ویژگی مهم (براساس معیار p-value) و پرتکرار استخراج می‌شوند. براساس این بردارهای زیر-ویژگی، گراف‌ها به داخل گروه‌های کوچکی تقسیم می‌شوند؛ به‌گونه‌ای که گراف‌های داخل یک گروه یکسان، بردارهای زیر-ویژگی مشابهی دارند. از شباهت بالای بردارهای موجود در یک گروه می‌توان به این نتیجه رسید که گراف‌های دربرگیرنده‌ی این بردارها نیز دارای شباهت بالایی هستند، و در نتیجه زیرگراف‌های با تکرار خیلی بالا را به اشتراک می‌گذارند. به همین دلیل، از یک راه‌کار کاوش زیرگراف پرتکرار با آستانه فراوانی بالا، برای استخراج زیرگراف‌های پرتکرار داخل آن گروه، استفاده می‌شود. به این ترتیب چون که از آستانه فراوانی بالا برای کاوش

1- Frequent Subgraph
2- Statistical Significance
3- Discrimination Score
4- Random Walk and Restart

زیرگراف استفاده شده است، از شمارش یک تعداد خیلی زیاد زیرگراف‌های نامزد، که منجر به ایجاد یک گلوگاه شدیدی در روش‌های ابتدایی می‌شد، پیشگیری شده است. این روش قادر است زیرگراف‌های مهم را به‌صورت کاملاً مؤثری استخراج کند. جزئیات بیشتر این روش در [۱۹] قابل مشاهده است.

۳-۴- شناسایی زیرگراف‌های متمایزکننده

تا به اینجا به‌ازای هر خانواده بدافزار، یک مجموعه زیرگراف مهم استخراج شده است. در این مرحله از روش پیشنهادی، یک کاوش مجدد بر روی هر یک از این مجموعه زیرگراف‌های مهم انجام می‌دهیم، تا به ازای هر خانواده بدافزار برخی از زیرگراف‌های متمایزکننده ارزشمند موجود در آن خانواده را شناسایی کنیم. این زیرگراف‌های متمایزکننده، در بر دارنده رفتارهای مخربی هستند که قابلیت تمیز دادن بدافزارها از برنامه‌های سالم را دارند.

به منظور تعیین زیرگراف‌های تفکیک‌پذیر، نیاز به تعریف یک تابع هدف برای امتیازدهی به زیرگراف‌ها براساس میزان فراوانی زیرگراف‌ها در مجموعه برنامه مخرب و مجموعه برنامه سالم است. به گونه‌ای که هر چقدر یک زیرگراف قابلیت ایجاد تفکیک‌پذیری بیشتری بین مجموعه مخرب و مجموعه سالم داشته باشد، امتیاز بیشتری خواهد داشت. با فرض این که $p(g)$ و$q(g)$ به ترتیب نرخ فراوانی زیرگراف g در مجموعه مخرب و مجموعه سالم باشند، تابع هدف برای امتیازدهی به زیرگراف g، یعنی $F(g)$، یک تابع از p و q (به ترتیب شکل خلاصه‌شده $p(g)$ و $q(g)$) است:

$$\boldsymbol{F(g) = f(p,q)} \qquad (۱)$$

به‌طور کلی اگر اختلاف نرخ فراوانی یک زیرگراف g در مجموعه مخرب و مجموعه سالم افزایش یابد، این زیرگراف قابلیت تفکیک‌پذیری بیشتری بین دو مجموعه ایجاد می‌کند. در نتیجه اگر اختلاف نرخ فراوانی‌ها نسبت به هر دو مجموعه مخرب و سالم ملاک تفکیک‌پذیری یک زیرگراف باشد، آنگاه تعریف فوق بایستی هر دو ویژگی زیر را داشته باشد:

$$\boldsymbol{if\ p > q,\ \frac{\partial F}{\partial p} > 0,\ \frac{\partial F}{\partial q} < 0} \qquad (۲)$$

$$\boldsymbol{if\ p < q,\ \frac{\partial F}{\partial p} < 0,\ \frac{\partial F}{\partial q} > 0} \qquad (۳)$$

که در این صورت، تعریف فوق توابع هدف زیادی از جمله آزمون G و بهره اطلاعاتی۱ را تحتِ پوشش قرار می‌دهد. هر دو تابع اشاره شده، به‌طور کلی آزمون‌های متقارن یا دو سویه هستند؛ به این معنی که اختلاف نرخ فراوانی نسبت به هر دو مجموعه مخرب و سالم مد نظر قرار داده می‌شود و در نتیجه در هر دو سمت (به ازای $p > q$ و $q > p$)، تابع شیب مثبتی را طی می‌کند (رابطه (۲) و (۳)). در این مقاله، یک تابع آزمون جدید به نام آزمون H، به‌منظور آزمون میزان تفکیک‌پذیری یک زیرگراف پیشنهاد شده است. برای توجیه دقیق این آزمون، در ادامه مروری بر آزمون G صورت گرفته است. آزمون G یک آزمون بر اساس نرخ درست‌نمایی با توزیع دوجمله‌ای است و بررسی می‌کند که «آیا نرخ فراوانی یک زیرگراف در مجموعه مخرب، با توزیع آن زیرگراف در مجموعه سالم سازگار است یا خیر» [۲۲]. فرمول محاسبه نیز در رابطه (۴) آمده است.

$$\boldsymbol{G_{test} = 2\ln\frac{\ell(p|n)}{\ell(q|n)} = 2m\left(p.\ln\frac{p}{q} + (1-\mathrm{p}).\ln\frac{(1-p)}{(1-q)}\right)} \qquad (۴)$$

در این تعریف m تعداد گراف‌های موجود در مجموعه مخرب است و $\ell(\mathrm{p|n})$ و $\ell(\mathrm{q|n})$ به ترتیب تابع درست‌نمایی بردار p و q با بردار مشاهدات n است.

$$\boldsymbol{\ell((p,1-p)|(n_1,n_2)) = k\,p^{n_1}(1-p)^{n_2}} \qquad (۵)$$

$$\boldsymbol{\ell((q,1-q)|(n_1,n_2)) = k\,q^{n_1}(1-q)^{n_2}} \qquad (۶)$$

با توجه به این که هدف ما، تنها شناسایی زیرگراف‌های متمایزکننده مخرب است، لذا می‌توان با حذف قسمت دوم رابطه (۴) و همچنین ضرایب ثابت آن، تابع آزمون تک‌سویه G را به‌صورت رابطه (۷) نوشت.

$$\boldsymbol{G_{test\ OneTail} = p.\ln\frac{p}{q}} \qquad (۷)$$

باکمی تأمل بر روی رابطه (۷) می‌توان دریافت در حالتی که نسبت لگاریتمی نرخ فراوانی مخرب و سالم برای دو زیرگراف g_1 و g_2 یکسان باشد، این تابع آزمون به زیرگرافی که نرخ فراوانی مخرب بزرگ‌تری داشته باشد امتیاز بیشتری خواهد داد. اما ارزیابی‌های ما همواره نشان داده‌اند، که در چنین وضعیتی انتخاب زیرگرافی که نرخ فراوانی کمتری در مجموعه سالم دارد، انتخاب بهتری است. در واقع، این انتخاب اعتبار خیلی بیشتر به زیرگراف‌هایی خواهد داد که به‌ندرت در مجموعه سالم پدیدار شده‌اند و به این ترتیب منجر به شناسایی زیرگراف‌هایی خواهد شد که ذات مخرب‌تری دارند. پس تابع هدف پیشنهاد شده، به‌منظور رتبه‌بندی زیرگراف‌های مهم، به صورت رابطه (۸) قابل تعریف است. این تابع را تابع آزمون H (برگرفته از Hypothesis testing) نامیده‌ایم.

$$\boldsymbol{F(g) = \mathrm{H_{test}} = \frac{1}{q}.\ln\frac{p}{q}} \qquad (۸)$$

باکمی محاسبه، مشاهده می‌کنیم که این تابع شرط رابطه (۲) را به‌درستی پوشش داده است. به این صورت که، اگر $p > q$ باشد آنگاه:

$$\frac{\partial \mathrm{F}}{\partial \mathrm{p}} = \frac{1}{q}.\frac{q}{p}.\frac{1}{q} = \frac{1}{pq} > 0$$

$$\frac{\partial \mathrm{F}}{\partial \mathrm{q}} = \left(-\frac{1}{q^2}.\ln\frac{p}{q}\right) + \left(\frac{1}{q}.\frac{q}{p}.\frac{-p}{q^2}\right) = -\left(\frac{1}{q^2}.\ln\frac{p}{q}\right) - \frac{1}{q^2} < 0$$

1- Information Gain

الگوریتم (۱). تعیین زیرگراف‌های متمایزکننده

DiscriminativeSubgraphsSelection(G, G^+,G^-, k)

Input: a set of significant subgraph G
Input: a set of melecious graph G^+
Input: a set of benign graph G^-
Input: a positive integer k in case of cutoff.k and a numeric between 0 and 1 in case of cutoff.k.percent
Output: a set of discriminative subgraph D

1. **scoretable** as a table containing ranks for subgraphs in the first column and their ids as row ids
2. $\boldsymbol{D} \leftarrow \emptyset$
3. **for each** $\boldsymbol{g} \in \boldsymbol{G}$ **do**
4. $\mathbf{p} = \frac{|\forall g_i \in G^+ | g \subseteq g_i|}{|G^+|}$
5. $\mathbf{q} = \frac{|\forall g_i \in G^- | g \subseteq g_i|}{|G^-|}$
6. $\mathbf{score} = \frac{1}{q}.ln\frac{p}{q}$
7. $scoretable(g.id, score)$
8. $\boldsymbol{D} \leftarrow cuttoff.k.percent(scoretable, k)$ Or $\boldsymbol{D} \leftarrow cuttoff.k(scoretable, k)$
9. **return** $\boldsymbol{D}$

در قسمت ارزیابی، مقایسه دقیقی را بین تابع آزمون H با تابع آزمون تک‌سویه G و بهره اطلاعاتی انجام داده‌ایم. به این ترتیب، اکنون آماده‌ایم تا از تابع آزمون G برای رتبه‌بندی قدرت تفکیک‌پذیری زیرگراف‌های هر مجموعه زیرگراف مهم، استفاده کنیم. سپس به‌ازای هر مجموعه زیرگراف مهم/به ازای هر خانواده، k زیرگراف با بیشترین امتیاز تفکیک‌پذیری به‌عنوان الگوهای متمایزکننده برای آن خانواده بدافزار انتخاب خواهند شد. البته با توجه به متفاوت بودن ذات هر خانواده بدافزار، از لحاظ گستره، عملکردهای متفاوتی که هر خانواده از خود نشان می‌دهد، انتخاب k یکسان برای همه خانواده بدافزار کار دشواری است. به همین منظور انتخاب k می‌تواند توسط شخص تحلیل‌گر بدافزار، که دارای شناخت کافی نسبت به ماهیت هر خانواده بدافزار است، به‌صورت دستی هم مشخص شود. ارزیابی‌های انجام شده، نشان می‌دهند که برای هر خانواده از بدافزار انتخاب k برابر با ۲۰٪ از اندازه مجموعه زیرگراف مهم استخراج شده برای آن خانواده، یک انتخاب شایسته است. به این ترتیب به ازای هر خانواده بدافزار، مجموعه مرجعی از الگوهای مخربی را خواهیم داشت که توانایی تفکیک بدافزارهای مربوط به این خانواده از برنامه‌های سالم را خواهند داشت. الگوریتم (۱)، جزئیات این مرحله در انتخاب الگوهای رفتاری مخرب، به ازای هر خانواده بدافزار را نشان می‌دهد. در این الگوریتم مجموعه زیرگراف مهم با G ، مجموعه گرافِ سالم با G^- و مجموعه گراف مخرب با G^+ نشان داده شده است. تابع cuttoff یک زیرمجموعه از زیرگراف‌های رتبه‌بندی شده را بر می‌گرداند. به این صورت که اگر k یک عدد صحیح باشد از تابع cutoff.k استفاده می‌شود که یک مجموعه با k زیرگراف با بهترین امتیاز را برمی‌گرداند. و اگر k یک عدد اعشار بین ۰ تا ۱ باشد از تابع cutoff.k.percent استفاده می‌شود که یک مجموعه با k درصد از زیرگراف‌های که بهترین امتیاز دارند، را بر می‌گرداند. در ضمن scoretable نیز یک جدول است که شناسه هر زیرگراف مهم، به‌صورت سطر و امتیاز مربوط به آن در اولین ستون جدول درج شده است.

۴- پیاده‌سازی

ما تلاش‌های زیادی را برای ایجاد یک ابزار نظاره‌گر دقیق، به‌منظور رهگیری تمام تعاملاتی که یک برنامه با سیستم‌عامل انجام می‌دهد، ترتیب داده‌ایم. این ابزار نظاره‌گر مبتنی بر رهگیری فراخوانی‌های سیستمی در سطح هسته است و به این ترتیب قادر به رهگیری تمام بدافزارها از جمله روت‌کیت‌ها (در سطح هسته اجرا می‌شوند) است. بر طبق نتایج مستند شده شرکت سایمَنتِک [۲۱]، در هر ماه به‌طور متوسط ۲۰٪ از بدافزارها اقدام به شناسایی محیط ماشین مجازی می‌کنند. به همین دلیل، ما از ماشین مجازی برای ردیابی رفتار بدافزار استفاده نکرده‌ایم، بلکه تمام بدافزارها را در یک محیط واقعی Windows 7، ۳۲ بیتی اجرا کرده‌ایم. پس از اجرای تعدادی بدافزار، با برگرداندن نسخه پشتیبان، سیستم عامل به حالت اولیه و تمیز برگشت داده می‌شود. به‌منظور تهیه و بازیابی نسخه پشتیبان از سطح کل دیسک از نرم‌افزار Norton.Gost، ۲۰۱۵، استفاده کرده‌ایم. گراف رفتار مبتنی بر وابستگی بین فراخوانی-های سیستمی با دنبال کردن زنجیره تعریف-استفاده[1] استخراج شده است. به این صورت که هر آرگومان خروجی (در براکت out) یک تعریف و هر آرگومان ورودی (در براکت in) یک استفاده است. البته دوره زنده ماندن هر تعریف، بسته به این که از چه نوعی باشد متفاوت است. به این صورت که در نوع handle پایان عمر یک تعریف، تنها به‌وسیله تابع NtClose مشخص می‌شود. ولی در دو نوع دیگر pvoid و pclient_id پایان عمر یک تعریف، یک تعریف مجدد از همان نوع با همان مقدار اشاره‌گر است.

۵- ارزیابی

برای ارزیابی کارایی روش پیشنهادی در استخراج الگوهای رفتاری مخرب، از پنج خانواده مختلف بدافزار که به‌وفور در اطراف ما یافت می‌شوند و یک مجموعه ۷۰ تایی از برنامه‌های سالم پرکاربرد، استفاده شده است. جدول (۱) اسامی برنامه‌های سالم استفاده شده در ارزیابی را نشان می‌دهد. این مجموعه شامل ۷۰

1- Def-Use Chain

جدول (۲). برنامه‌های سالم استفاده شده در ارزیابی

AcrobatAcrobat Pro.11 setup	BitTorrent	Internet explore	KMPlayer	Notepad	Sublime.Text.3 Build.3012	Ultra Uninstaller
Acrobat Acrobat Pro.11	BitTorrent Setup	JetAudio	KMPlayer uninstall	Notepad++	Sublime.Text.3.Build .3012 Extracting	uTorrent
Acronis Backup & Recovery 11.5 Setup	Active Boot Disk 10.0.1 Suite	jetChat	Lingoes	Notepad uninstall	sublime_text1	uTorrent Setup
Acronis-Management Console	calc	JetLogo	Lingoes Setup	npp.6.5.3.Installer	Sublime uninstaller	VMware Workstation v.8
Windows Media Player	Windows Media Center Shell	JetLyric	Lingoes uninstaller	nslookup	Jet Uninstaller	WinRAR
Active File Recovery Pro setup	File Recovery	JetRecorder	Magnify	opera	Total Video Converter	WinRAR Setup
Active File Recovery	Firefox Setup Stub 39.0	JetShell	Media Player Calssic Setup	Opera_1202_int Setup	Total Video Player	WinRAR Uninstaller
Adobe Dreamweaver CC	Internet Download Manager	JetTrim	Media Player Calssic UnInstaller	Active@ Partition Recovery	Total Video Uninstaller	Windows Update
Adobe Dreamweaver CC setup	Internet Download Manager Setup	jetUpdate	Microsaft Paint	SnippingTool	UltraISO	Total Video Setup
Babylon	Internet Download Manager Uninstall	kmplayer 3.9 Setup	Remote Desktop Connection	StikyNot	UltraISO setup	Media Player Clsassic

برنامه سالم با کاربردهای مختلف، شامل برنامه‌های دسترسی به وب، مرورگرها، برنامه‌های دیسک، برنامه‌های بازیابی، دانلودر[1]ها، نصب‌کننده‌ها[2]، جداکننده‌ها[3]، فشرده‌کننده‌ها و غیره استفاده شده است. همان‌طور که می‌دانید برنامه‌هایی مثل نصب‌کننده‌ها و جداکننده‌ها از آن جا که یک مجموعه عملیات در سطح امتیاز مدیر (از جمله تعبیه یک تکه کد برای اجرا شدن در هر بار راه‌اندازی مجدد سیستم و ایجاد تغییر در تنظیمات پیکربندی فعلی سیستم) انجام می‌دهند، به‌شدت به رفتارهای مخرب که بدافزارها از خود نشان می‌دهند، نزدیک هستند. سهم زیادی از مجموعه انتخابی ما به برنامه‌های نصب‌کننده و جداکننده اختصاص داده شده است. با توجه به این که اکثر عملکردهای موجود در برنامه‌های سالم پنهان هستند، تعامل با هر برنامه به‌صورت دستی صورت گرفته است تا تمام عملکردهای که در یک برنامه دارد، حداقل یک بار اجرا شوند.

جدول (۱). خانواده‌های مختلف بدافزار استفاده‌شده در ارزیابی

نام خانواده	توصیف
Bagle	کرم[4]، تکثیر از راه پست الکترونیک
Virut	ویروس، تخریب فایل‌ها
Fesber	کرم، تکثیر از راه شبکه
Delf	کرم، تکثیر از راه شبکه
Poison	درب پشتی[5]

1- Downloader
2-Installer
3- Uninstaller

۴- کرم یا Worm، این دسته از بدافزار ممکن است اهداف مختلفی در سطح دیگر بدافزارها داشته باشند. ولی ویژگی اصلی آن‌ها تکثیر است. این تکثیر می‌تواند از طریق شبکه محلی یا اینترنت، پستِ الکترونیک (Email) با الصاق فایل مخرب، شبکه نظیر به نظیر، بلوتوث و غیره برای آلوده کردن سیستم قربانی‌ها انجام شود.

۵- دربِ پشتی یا Backdoor، این دسته از بدافزار تمایل به دسترسی و دستکاری برنامه‌ها، شبکه و غیره از راه دور دارند.

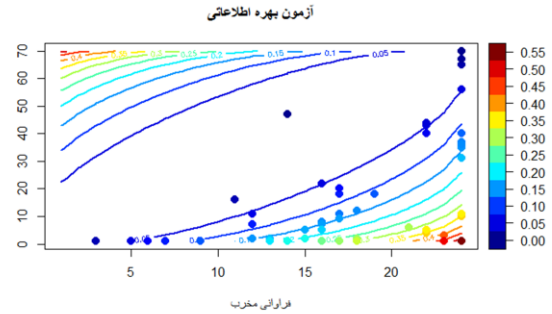
جدول (۲)، ویژگی‌های مربوط به پنج خانواده بدافزار استفاده شده در ارزیابی را نشان می‌دهد. رفتار هر بدافزار به‌طور متوسط ۳ دقیقه تحت نظر قرار گرفته است. این مدت زمان برای مشاهده عملکرد بدافزارها کافی است. از آنجا که برخی بدافزارها برای اطمینان از عدم حضور خود در یک محیط تحلیل نیاز دارند که وجود کاربر را همچنان احساس کنند، لذا در ضمن تحلیل، تعامل با سیستم‌عامل همچنان حفظ شده است.

منبع اصلی ما برای جمع‌آوری بدافزارها سایت VirusShare.com بوده است. این سایت از سال ۲۰۱۲ تاکنون، تقریباً در بازه‌های زمانی ۶۰ روزه، اقدام به انتشار بدافزارهای جدید، در قالب یک فایل تورنت می‌کند. علاوه بر این، به‌منظور دسته‌بندی بدافزارها بر اساس عملکرد، از گزارشات ضد-بدافزار کسپرسکی، نسخه ۲۰۱۵، استفاده شده است. پیش از ارزیابی و مقایسه الگوهای رفتاری مخرب استخراج شده توسط روش پیشنهادی ، ابتدا نتایج حاصل از تابع آزمون پیشنهادی برای رتبه‌بندی زیرگراف‌های مهم مورد ارزیابی قرار گرفته است.

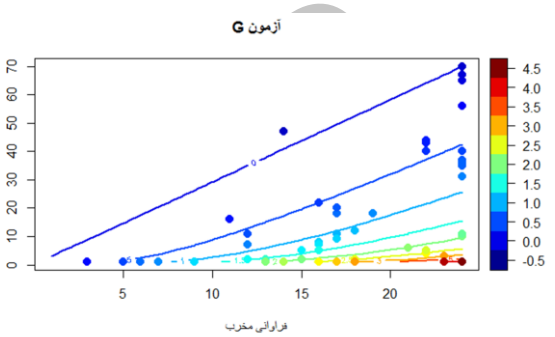
۵-۱- ارزیابی تابع آزمون H

برای ارزیابی تابع آزمون H، مقایسه‌ای بین نحوه‌رتبه‌بندی این تابع و توابع آزمون G و بهره اطلاعاتی صورت داده‌ایم. رتبه‌بندی بر روی یک مجموعه با ۷۰ زیرگراف مهم که از خانواده Bagle با ۲۴ بدافزار استخراج شده، انجام شده است. مجموعه سالم ما نیز شامل ۷۰ برنامه سالم فهرست شده در جدول (۱) است. نمودار مربوط به نحوه رتبه‌بندی تابع بهره اطلاعاتی، آزمون G و آزمون H به ترتیب در شکل ۱، ۲ و ۳ نشان داده است.

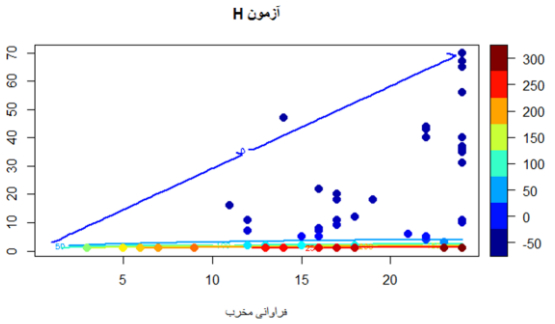
در این اشکال محور x، فراوانی زیرگراف در مجموعه گراف مخرب و محور y، فراوانی زیرگراف در مجموعه گراف سالم را نشان می‌دهد. هر یک از دایره‌های داخل نمودار، مربوط به یک زیرگراف است و رنگ آن گویای مقدار امتیازی است که تابع



شکل (۱). رتبه‌بندی ۷۰ زیرگراف مهم، استخراج شده از خانواده Bagle، بر اساس بهره اطلاعاتی



شکل (۲). رتبه‌بندی ۷۰ زیرگراف مهم، استخراج شده از خانواده Bagle، بر اساس آزمون G



شکل (۳). رتبه‌بندی ۷۰ زیرگراف مهم، استخراج شده از خانواده Bagle، بر اساس آزمون پیشنهادی H

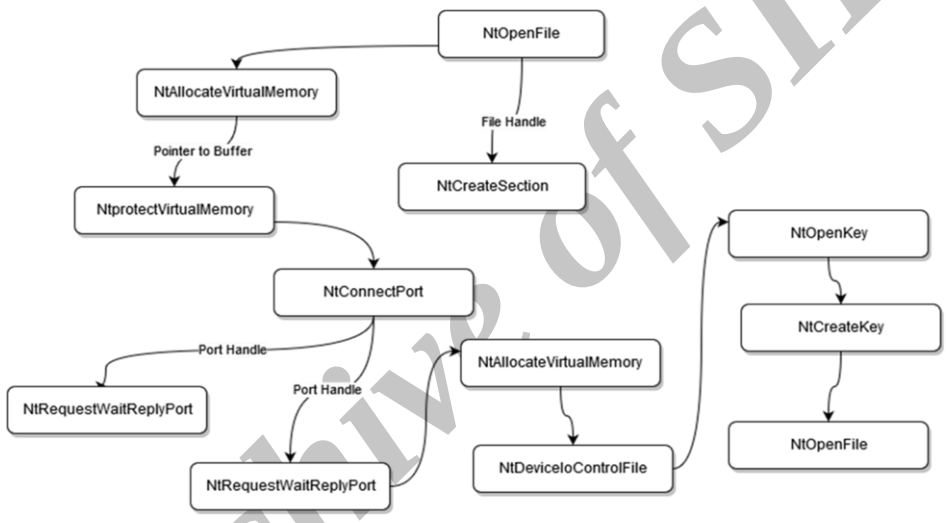
هدف، به آن زیرگراف داده است. به این صورت که، آبی کدر کمترین و قرمز کدر بیشترین امتیاز را نشان می‌دهند. به ازای هر رنگ مشخص‌شده در ستون سمت راست نمودار، یک منحنی در داخل نمودار وجود دارد که نحوه امتیازدهی به یک زیرگراف را با توجه به تغییرات مقدار فراوانی در مجموعه سالم و مخرب نشان می‌دهد. همان‌طور که از این نمودارها پیدا است که تابع آزمون H، توجه خیلی زیادی به پایین نگه‌داشتن نرخ فراوانی در مجموعه سالم دارد. به‌عنوان مثال تابع آزمون G انتخاب زیرگرافی با مقدار فراوانی مخرب ۲۳ و مقدار فراوانی سالم ۲ را به انتخاب زیرگرافی که فراوانی مخرب آن ۱۷ و مقدار فراوانی سالم آن صفر است، ترجیح می‌دهد. به همین ترتیب تابع بهره‌وری اطلاعاتی نیز همین زیرگراف را با انتخاب یک زیرگراف خیلی بدتری، که مقدار فراوانی مخرب آن ۲۴ و مقدار فراوانی سالم آن ۱۰ است، پس می‌زند. همه این‌ها باعث شده‌اند که از تابع آزمون پیشنهادی H برای انتخاب برترین زیرگراف‌ها استفاده شود.

۵-۲- ارزیابی و مقایسه الگوهای رفتاری مخرب استخراج شده

نتایج کلی حاصل از ارزیابی‌های انجام شده در جدول (۳) ارائه شده است. ستون دوم این جدول، تعداد بدافزارهای انتخاب شده به ازای هر خانواده را نشان می‌دهد. لازم به ذکر است که اندازه مجموعه سالم در نظر گرفته شده نیز ۷۰ است. ستون سوم جدول تعداد زیرگراف‌های مهم و غیر متشابه استخراج شده توسط الگوریتمِ کاوش زیرگراف‌های مهم، به ازای هر خانواده را نشان می‌دهد. ستون چهارم جدول، تعداد الگوهای رفتاری مخرب متفاوت انتخاب شده به ازای هر خانواده را نشان می‌دهد. این تعداد برابر با ۲۰٪ از زیرگراف‌های مهم برتر انتخاب‌شده براساس تابع آزمون H است که بیشترین قابلیت تمیز دهندگی مجموعه گراف مخرب از سالم را دارند. نرخ پوششی که این الگوها در شناسایی مجموعه مخرب انتخاب شده به ازای هر خانواده بدافزار و مجموعه سالم ایجاد می‌کنند، به ترتیب در ستون‌های پنجم و ششم دیده می‌شود. همان‌طور که مشاهده می‌کنید، علیرغم مجموعه گسترده و متنوع برنامه‌های سالم، نرخ خطای کاذب برای تمام الگوهای مخرب انتخاب‌شده به ازای هر خانواده کاملاً صفر است.

جدول (۳). نتایج حاصل از استخراج الگوهای رفتاری مخرب به ازای پنج خانواده بدافزار با در نظر گرفتن ۷۰ برنامه سالم

مرحله یادگیری						مرحله آزمون	
خانواده بدافزار	اندازه مجموعه مخرب	تعداد زیرگراف‌های مهم متمایز	تعداد الگوهای رفتاری مخرب براساس k (۲۰٪ برتر)	نرخ پوشش در مجموعه مخرب	نرخ پوشش در مجموعه سالم	اندازه مجموعه آزمون مخرب	دقت شناسایی
Bagle	۲۴	۷۰	۱۵	٪۱۰۰	٪۰	۲۰	۱/۰۰
Virut	۴۰	۷۸	۱۵	٪۱۰۰	٪۰	۳۰	۰/۹۶
Fesber	۳۰	۷۰	۲۶	٪۱۰۰	٪۰	۲۰	۱,۰۰
Delf	۳۰	۷۹	۱۹	۱۰۰٪	۰٪	۲۰	۰/۹۰
PoisonIvy	۴۰	۸۵	۲۶	٪۱۰۰	٪۰	۳۰	۰/۸۶
مجموع/میانگین	۱۶۴	۳۸۲	۱۰۱	٪۱۰۰	٪۰	۱۱۰	**۰/۹۴**



انتخاب k: همان‌طور که پیش از این نیز مطرح شده است، با توجه به اینکه هر خانواده بدافزار گستره عملکردی کاملاً متفاوتی نسبت به یکدیگر دارند، انتخاب یک k یکسان برای همه خانواده‌ها دشوار است. در واقع، تعیین این که تا چه میزان دقت می‌توان پنداشت که یک الگو شامل یک نیت مخرب است، یک چالش بزرگ در کار ما است. مثلاً در ارزیابی‌های صورت گرفته، مشاهده شده است که انتخاب حتی عددی بیشتر از ۲۰٪ از زیرگراف‌ها برای خانواده Bagle و Fesber نیز می‌تواند همچنان شامل زیرگراف‌های با گرایش مخرب باشد؛ به طوری که برای تمام ۲۶ زیرگراف استخراج شده، نرخ فراوانی مجموعه مخرب ۱۰۰٪ و نرخ فراوانی مجموعه سالم ۰٪ است. برای خانواده Bagle نیز حتی آخرین الگوهای استخراج شده دارای نرخ فراوانی مخرب نزدیک به ۸۰٪ و نرخ فراوانی سالم آن‌ها ۰ است؛ و بالعکس در خانواده Delf مشاهده شده است که انتخاب ۲۰٪ از زیرگراف‌ها، تقریباً زیاد است و منجر به تعیین زیرگراف‌های می‌شود که از میزان دقت کافی برخوردار نیستند. اساساً تعیین k می‌تواند توسط شخصِ تحلیلگر، که به ویژگی‌های عملکردی هر خانواده آشنا است، با دقت بهتری انتخاب شود، اما به‌هرحال ما در ارزیابی‌ها، k را برابر با ۲۰ درصد از اندازه مجموعه زیرگراف‌های مهم انتخاب کرده‌ایم.

- **میزان دقت در کشف بدافزار**

به‌منظور ارزیابی دقت الگوهای رفتاری مخرب استخراج شده، به ازای هر خانواده یک مجموعه آزمون شامل تعدادی بدافزار در نظر گرفته شده است. این تعداد در ستون هفتم جدول (۳) مشخص شده است. نتایج حاصل از میزان دقت الگوهای رفتاری مخرب در شناسایی این مجموعه آزمون در ستون هشتم این جدول نشان داده شده است. همان‌طور که مشاهده می‌کنید، ما توانسته‌ایم بدافزارهای دو خانواده Bagle و Fesber را با دقت کامل ۱۰۰٪ شناسایی کنیم. اما در مابقی خانواده موفق به شناسایی تعداد اندکی از بدافزارهای مجموعه آزمون نشده‌ایم. به‌طور کلی، راه‌کار ما توانسته است الگوهای رفتاری مخرب متمایزکننده موجود در هر خانواده بدافزار که قابلیت تمیز دادن برنامه‌های مخرب از برنامه‌های سالم را دارند، را استخراج کند و

به‌دقت ۹۴٪ در شناسایی بدافزارهای ناشناخته با نرخ خطای کاذب صفر دست یابد که در مقایسه با نرخ کشف ۵۵٪ در ضد بدافزارهای تجاری مبتنی بر امضاء، نرخ کشف ۶۴٪ در کشف‌کننده رفتاری ارائه شده توسط Kolbitsch و همکاران [۱۳] و همچنین نرخ کشف ۸۶٪ در بهترین کشف‌کننده‌ی رفتاری ارائه شده تاکنون (Fredrikson و همکاران [۱۴])، توانسته‌ایم به یک بهبود قابل توجهی دست یابیم.

- **مقایسه با رفتارهای گزارش‌شده توسط ضد- بدافزارهای تجاری**

به‌منظور مقایسه الگوهای رفتاری استخراج‌شده توسط روش پیشنهادی با رفتارهای مخربِ گزارش‌شده توسط تحلیل‌گرهای حرفه‌ای، از گزارش‌های ارائه شده توسط دو ضد-بدافزار تجاری کسپرسکی [۲۲] و F-Secure [۲۳] استفاده شده است. بر طبق این گزارش‌ها، تقریباً اکثر الگوهای رفتاری استخراج شده توسط روش پیشنهادی، دقیقاً با توصیف ارائه‌شده برای آن خانواده همخوانی داشته است. بر طبق گزارش ارائه‌شده توسط F-Secure برای خانواده PoisonIvy، برخی از بدافزارهای این دسته پیش از ایجاد یک اتصال خارجی، اقدام به تزریق کد بدخواه خود به داخل مرورگرها برای دور زدن دیواره آتش[1] ویندوز می‌کنند. زیرگراف مربوط به این رفتار به‌درستی در تقریباً در نیمی از مجموعه بدافزارهای خانواده PoisonIvy دیده شده است. به‌طوری که عمل تزریق کد بدخواه خود را به داخل مرورگر ixplore.exe انجام داده‌اند.

برای خانواده Bagle، اولین یا برترین زیرگراف مخرب استخراج‌شده، رفتار مربوط به نحوه‌ی تکثیر یک بدافزار از نوع کرم را نشان می‌دهد. همان‌طور که در الگوی رفتاری مخرب شکل (۴) نیز قابل مشاهده است، بدافزارهای مربوط به خانواده Bagle ابتدا به‌وسیله تابع NtOpenFile، فایل اجرایی نسخه اصلی خودش را از روی دیسک باز کرده و به‌وسیله تابع NtCreateSection، تمام قسمت کد خود را به داخل حافظه منتقل می‌کند. در واقع با این کار قادر خواهد بود که از این پس اجرای خودش را از داخل حافظه دنبال می‌کند. بارها دیده شده است که بدافزارها، پس از ایجاد یک نسخه از خود در حافظه، به‌سرعت فایل اجرایی خود را از روی دیسک، به‌منظور مخفی ماندن هر چه بیشتر، حذف کرده‌اند. در ادامه اجرای خود، اقدام با ایجاد یک اتصال به بیرون از سیستم کرده است (تابع NtConnectPort) تا به این صورت فایل ایجاد شده از خودش را که اکنون در حافظه قرار دارد، به بیرون از سیستم انتشار دهد.

- **ارزیابی دقت الگوهای رفتاری مخرب**

در این قسمت، به‌عنوان نمونه مشخصات دقیق زیرگراف‌های استخراج‌شده به ازای خانواده Bagle را در جدول (۴)، ارائه کرده‌ایم. این جدول شامل ۲۰٪ برتر از زیرگراف‌های مهم، بر اساس امتیازدهی تابع آزمون H است. برای این خانواده یک مجموعه آزمون که شامل ۲۰ بدافزار جدید از این خانواده است، ایجاد شده است و دقت هر زیرگراف براساس میزان پوششی که این زیرگراف در مجموعه آزمون ایجاد کرده است، محاسبه شده است. همان‌طور که مشاهده می‌کنید، حتی آخرین الگوهای رفتاری مخرب استخراج شده برای خانواده Bagle، نیز از دقت قابل قبولی در شناسایی بدافزارهای مجموعه Bagle، با نرخ خطای کاذب صفر برای ۷۰ برنامه متنوع سالم، برخوردار هستند.

۶- محدودیت‌ها و کارهای آتی

اگرچه که ما توانسته‌ایم به ازای هر یک از خانواده بدافزارهای ارزیابی‌شده الگوهای را استخراج کنیم که قابلیت تمیز دادن مجموعه مخرب از مجموعه برنامه‌های سالم را دارند ولی

جدول (۴). دقت الگوهای رفتاری مخرب استخراج‌شده برای خانواده Bagle با مجموعه آزمون شامل ۲۰ بدافزار

ردیف	نرخ فراوانی مخرب	نرخ فراوانی سالم (خطای کاذب)	نرخ فراوانی در مجموعه آزمون (میزان دقت)
۱	٪۱۰۰	٪۰	٪۱۰۰
۲	٪۱۰۰	٪۰	٪۱۰۰
۳	٪۱۰۰	٪۰	٪۱۰۰
۴	٪۱۰۰	٪۰	٪۱۰۰
۵	٪۱۰۰	٪۰	٪۱۰۰
۶	٪۱۰۰	٪۰	٪۱۰۰
۷	٪۱۰۰	٪۰	٪۱۰۰
۸	٪۱۰۰	٪۰	٪۱۰۰
۹	٪۹۵/۸۳	٪۰	٪۱۰۰
۱۰	٪۷۵	٪۰	٪۸۰
۱۱	٪۷۵	٪۰	٪۸۰
۱۲	٪۷۵	٪۰	٪۸۰
۱۳	٪۷۵	٪۰	٪۸۰
۱۴	٪۷۵	٪۰	٪۸۰
۱۵	٪۷۵	٪۰	٪۸۰

نتوانسته‌ایم آن‌ها را به‌عنوان نقش‌های معنادار مخرب به‌گونه‌ای توصیف کنیم که برای شخص تحلیل‌گر نیز به‌خوبی قابل فهم باشند. در واقع به نظر می‌رسد که نیاز به انجام یک تحلیل معنایی پیچیده بر روی الگوهای مخرب استخراج شده به‌ازای هر خانواده بدافزار وجود دارد تا بتوانیم نقش‌های[2] مخرب موجود در

1 -Firewall

2 -Rules

یک خانواده را به شکلی توصیف کنیم که قابلیت فهم بیشتری داشته باشند. یکی از ملزومات پرداختن به این موضوع، در نظر گرفتن جزئیات بیشتر در دسترسی به منابع و اشیاء حساس موجود در سیستم‌عامل است. منابع حساس سیستم‌عامل شامل منابعی است که با امنیت، پایداری و کارایی سیستم‌عامل در ارتباط هستند. مدیر وظایف[1]، راه‌اندازی سیستم[2]، تنظیمات دیواره آتش و فایل‌های اجرایی مربوط به خود سیستم‌عامل، از جمله منابع خیلی حساس موجود در سیستم‌عامل ویندوز هستند که اکثر بدافزارها اقدام به دسترسی و دست‌کاری آن‌ها می‌کنند. به این ترتیب با در نظر گرفتن دسترسی به منابع حساس در تشکیل گراف رفتار برنامه، به توصیف دقیق‌تری از فعالیت‌های مخرب دست خواهیم یافت.

۷- نتیجه‌گیری

نتایج ارزیابی‌های ما همواره نشان داده است که روش پیشنهادی توانسته است به‌ازای هر یک از خانواده‌های ارزیابی‌شده، تقریباً تمام رفتارهای مخرب شایع گزارش شده توسط ضد بدافزارهای تجاری را به‌درستی شناسایی کند. علاوه بر این در ارزیابی‌ها نشان داده شده است که برترین زیرگراف انتخاب‌شده به‌ازای هر یک از خانواده‌های Bagle، Virut، Fesber توانسته است، نرخ پوشش ۱۰۰٪ را برای مجموعه مخرب و آزمون ایجاد کنند. ما توانسته‌ایم به‌دقت ۹۴ درصدی در کشف بدافزار مبتنی بر رفتار دست یابیم که در مقایسه با کارهای مشابه (Kolbitsch [۱۳] با دقت ۶۲٪ و Fredrikson [۱۴] با دقت ۸۶٪) به یک دستاورد مهم دست یافته‌ایم.

۸- مراجع

[1] C. Funk and M. Garnaeva, "Kaspersky Security Bulletin 2013," Kaspersky Lab, 2013.

[2] C. Fred, "Computer viruses: Theory and experiments," Computers & Security, vol. 6, no. 1, pp. 22-35, 1987.

[3] V. Nair, H. Jain, and Y. Golecha, "MEDUSA: MEtamorphic malware dynamic analysis usingsignature from API," Proceedings of the 3rd international conference on Security of information and networks, ACM, 2010.

[4] P. O'Kane, S. Sezer, and K. McLaughlin, "Obfuscation: The Hidden Malware," IEEE Symposium on Security & Privacy, Oakland, 2011.

[5] P. Szor, "The art of computer virus research and defense," Pearson Education, 2005.

[6] M. Egele, T. Scholte, E. Kirda, and C. Kruegle, "A survey on automated dynamic malware-analysis techniques and tools," CSUR, vol. 44, no. 2, pp. 1-42, 2012.

[7] C. Mihai, S. Seshia, and J. Somesh, "Semantics-aware malware detection," IEEE Symposium on Security and Privacy, Oakland, 2005.

[8] C. Kruegel, W. Robertson, and G. Vigna, "Detecting Kernel-Level Rootkits Through Binary Analysis," 20th Annual Conference on Computer Security Applications, Tucson, 2004.

[9] M. Andreas, C. Kruegel, and E. Kirda, "Limits of Static Analysis for Malware Detection," 23th Annual Conference on Computer Security Applications, Miami Beach, 2007.

[10] M. Suenaga, "A Museum of API Obfuscation on Win32," Proceedings of 12th Association of Anti-Virus Asia Researchers International Conference, Kyoto, 2009.

[11] S. Motevasel, H. Shirazi, and M. Farshchieyan, "Providing an efficient system for malware detection and classification," 1st National E-Conference of Technology Departments on Electrical, Electronics and Computer Engineering, Khayyam University, Mashhad, 2014.

[12] M. Lajevardi, S. Parsa, and M. Kangavari, "The design and implementation of a behavioral based malware detection," Master of Thesis in Computer Engineering School, Iran University of Science and Technology, Tehran, 2013.

[13] C. Kolbitsch and P. M. Comparetti, "Effective and Efficient Malware Detection at the End Host," USENIX security symposium, San Diego, 2009.

[14] J. Somesh, M. Fredrikson, and M. Christodore, "Synthesizing near-optimal malware specifications from suspicious behaviors," 8th International Conference on Malicious and Unwanted Software: The Americas (MALWARE), Fajardo, 2013.

[15] M. Bailey, J. Oberheide, J. Andersen, Z. M. Morley, F. Jahanian, and J. Nazario, "Automated Classification and Analysis of Internet Malware," International Workshop on Recent Advances in Intrusion Detection, Gold Coast, 2007.

[16] K. Rieck, T. Holz, C. Willems, P. Düssel, and P. Laskov, "Learning and Classification of Malware Behavior," International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, Paris, 2008.

[17] J. Kinable and O. Kostakis, "Malware classification based on call graph clustering," Computer Virology, vol. 7, no. 4, pp. 233-245, 2011.

[18] Y. Park, D. Reeves, V. Mulukutla, and B. Sundaravel, "Fast malware classification by automated behavioral graph matching," Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research, New York, 2010.

[19] S. Ranu and A. Singh, "Graphsig: A scalable approach to mining significant subgraphs in large graph databases," IEEE International Conference on Data Engineering, Shanghai, 2009.

[20] R. Sokal and F. J. Rohlf, "Biometry: the principles andpractice of statistics in biological research," San Francisco, 1995.

[21] C. Wueest, "Does malware still detect virtual machines?," Symantec Official Blog, 2014.

[22] Kaspersky Lab, "Malware Classifications," Kaspersky, [Online]. Available: http://www.kaspersky.com/internet-security-center/threats/malware-classifications.

[23] "Classification How F-Secure classifies threats," F-secure, [Online]. Available: https://www.f-secure.com/en/web/labs_global/classification.

1 -Task Manager

2 -System Startup

۹-پیوست

اسامی ۴۲ تابع رهگیری‌شده توسط راه‌انداز ProcessHooking.sys به همراه آرگومان‌های مورد استفاده در تشکیل گراف وابستگی

نام تابع	آرگومان‌های مهم	نام تابع	آرگومان‌های مهم
NtTerminateProcess	handle[in]	**NtFsControlFile**	handle[in]
NtLoadDriver	-	**NtMakeTemporaryObject**	handle[in]
NtOpenProcess	handle[out], pclient_id[in]	**NtOpenSection**	handle[out]
NtOpenFile	handle[out]	**NtProtectVirtualMemory**	handle[in], pvoid[in]
NtCreateFile	handle[out], pvoid[in]	**NtOpenThread**	handle[out], pclient_id[in]
NtOpenKey	handle[out]	**NtQueueApcThread**	handle[in]
NtClose	handle[in]	**NtReplaceKey**	handle[in]
NtCreateProcess	handle[out], handle[in], handle[in], handle[in], handle[in]	**NtRequestPort**	handle[in]
NtCreateProcessEx	handle[out], handle[in], handle[in], handle[in], handle[in]	**NtRequestWaitReplyPort**	handle[in]
NtCreateThread	handle[out], handle[in], pclient_id[out], pvoid[in]	**NtRestoreKey**	handle[in], handle[in]
NtCreateThreadEx	handle[out], handle[in], pvoid[in]	**NtSecureConnectPort**	handle[out]
NtAllocateVirtualMemory	handle[in], pvoid[out]	**NtSetContextThread**	handle[in]
NtAlpcCreatePort	handle[out]	**NtSetSecurityObject**	handle[in]
NtAlpcConnectPort	handle[out]	**NtSetSystemInformation**	-
NtAlpcSendWaitReceivePort	handle[in]	**NtShutdownSystem**	-
NtAssignProcessToJobObject	handle[in], handle[in]	**NtSuspendProcess**	handle[in]
NtConnectPort	handle[out]	**NtSuspendThread**	handle[in]
NtCreateKey	handle[out]	**NtSystemDebugControl**	pvoid[in], pvoid[out]
NtCreateSection	handle[out], handle[in]	**NtTerminateThread**	handle[in]
NtDeviceIoControlFile	handle[in], handle[in], pvoid[in], pvoid[out]	**NtUnloadDriver**	-
NtDuplicateObject	handle[in], handle[in], handle[in], handle[out]	**NtWriteVirtualMemory**	handle[in], pvoid[in], pvoid[in]

Providing a New Approach to Discovering Malware Behavioral Patterns Based on the Dependency Graph Between System Calls

S. Parsa*, H. Saifi, M. H. Alaeian

*Iran University of Science and Technology

(Received: 13/02/2016, Accepted: 01/08/2016)

ABSTRACT

Most malware producers use obfuscation techniques to bypass signature-based detections. In order to provide proactive and real-time protection, the researchers have begun to develop strategies for behavior-based detection. While behavior-based detection techniques are promising solutions to this growing problem of malwares varieties, but they still suffer from high false positive rate in detecting unknown malware detection. To overcome this problem, we shall seek for identifying patterns, representing malicious intent in all instances of a malware family. In this paper, we propose a new technique, based on discriminative subgraph mining technique to identify discriminative behavioral patterns. The malicious behavioral patterns discovered by our technique from a known malware set allows the detector to reach an 94% detection rate over unknown malware with no false positives. This is a significant improvement over the 55% detection rate observed from commercial antivirus, and the 86% rate reported by the best behavior-based detector.

Keywords: Behavior Graph, System Call Dependency Graph, Malicious Behavioral Pattern, Discriminative Subgraphs, Significant Subgraph Mining

* Corresponding Author Email: parsa@iust.ac.ir