نشریه علمی «علوم و فناوری‌های پدافند نوین»
سال دهم، شماره ۴، زمستان ۱۳۹۸؛ ص ۴۰۶-۳۹۳

ارائه راه‌کاری برای تشخیص زودهنگام و خنثی‌سازی حملات تزریق کد و کتابخانه در بدافزارها

دانیال جواهری[۱]، مهدی حسین‌زاده[۲*]

۱ – دکتری مهندسی کامپیوتر، ۲– دانشیار گروه مهندسی کامپیوتر، دانشگاه آزاد اسلامی، واحد علوم و تحقیقات، تهران، ایران.
(دریافت: ۹۷/۰۸/۰۷، پذیرش: ۹۷/۱۰/۲۶)

چکیده

آهنگ رشد بدافزارها در سال‌های اخیر به‌صورت فزآینده‌ای افزایش یافته است. همچنین رفتار بدافزارهای جدید در حال مبهم‌تر شدن و پیچیده‌تر شدن است. این مقاله ضمن تشریح روش‌های موجود برای تشخیص بدافزار به‌صورت خاص بر روی تشخیص زودهنگام حملات تزریق کد و کتابخانه متمرکز شده است. بدافزارهای نوین با تزریق کد بدخواه در فایل باینری و یا حافظه اجرایی برنامه‌های مجاز سعی در مبهم‌سازی و مخفی‌سازی رفتار خود دارند. روش پیشنهادی این مقاله با داده‌کاوی در حجم انبوه بدافزار، زنجیره فراخوانی‌های رفتار مخرب تزریق کد/کتابخانه را به‌وسیله نصب قلاب‌های شنودگر در فضای هسته سیستم‌عامل استخراج و بر اساس تابع رگرسیون خطی مدل‌سازی می‌کند. روش پیشنهادی برای تشخیص زود هنگام حمله از یادگیری مبتنی بر قواعد انجمنی بر اساس الگوریتم Apriori استفاده می‌کند و قادر است حملات را قبل از کامل شدن و از بین رفتن کنترل جریان اجرایی برنامه قربانی تشخیص دهد. همچنین روش پیشنهادی می‌تواند از وقوع حمله با انسداد فراخوانی ایجاد نخ راه دور جلوگیری کند. در انتها این مقاله دقت روش پیشنهادی خود در تشخیص بدافزارهای کلاس تزریق‌کننده را با مجموعه داده جمع‌آوری‌شده از مراجع معتبر ارزیابی و در شرایط یکسان با ابزارهای ضدبدافزار موجود مقایسه می‌کند. نتایج ارزیابی نشان می‌دهد که روش پیشنهادی می‌تواند با دقت نزدیک به ۹۴٪ حملات تزریق کد/کتابخانه را تشخیص دهد. همچنین ضریب موفقیت سامانه خود حفاظتی پیشنهادی در مواجهه با حملات تزریق کد/کتابخانه ۸۸/۸۸ سنجش شده است.

کلیدواژه ها: تحلیل بدافزار، کشف جاسوس افزار، تزریق کد، خود حفاظتی، مبهم‌سازی، مخفی‌سازی

A Solution for Early Detection and Negation of Code and DLL Injection Attacks of Malwares

D. Javaheri, M. Hosseinzadeh[*]
Department of Computer Engineering, Science and Research Branch, Islamic Azad University, Tehran, Iran.
(Received: 29/10/2018; Accepted: 16/01/2019)

Abstract

Malwares have grown drastically in recent years. Furthermore, the behavior of the newly produced malwares are getting more complex and shrewd. This paper present malware detection methods and especially focus on code and DLL injection attacks. Novel malwares try to obfuscate and hide their behavior through the injection of malicious code in allocated memory and binary file of trusted applications. By data mining on massive volume of malwares, the proposed method of the paper derive chain of API calls through installing logger hook at the kernel space of the operating system in order to model the malicious behavior of code/DLL injection based on linear regression function. The proposed method use association rules machine learning based on Apriori algorithm for early detection of attacks and is able to prevent completion of the attack by blocking remote thread creation. Finnaly, the accuracy of the proposed method is evaluated using dataset from valid references and the results are compared with available Antivirus tools under the same conditions. Results of the evaluation indicate that the proposed method can recognize code/DLL injection attacks by the accuracy of about 94%. Moreover, success coefficient of the proposed self-defense system is evaluated of 88.88% against real code/DLL injection attacks.

Keywords: Malware Analysis, Spyware Detection, Code Injection, Self-defense, Obfuscation, Stealth

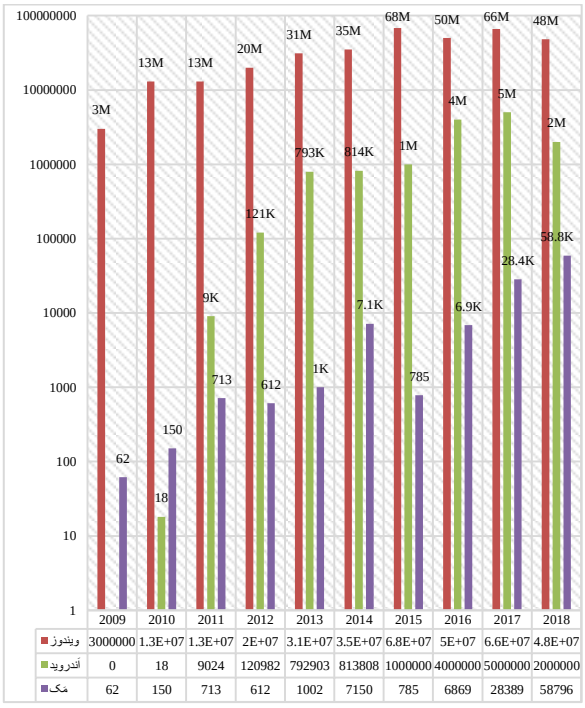
[*] Corresponding Author Email: hosseinzadeh@srbiau.ac.ir

۱. مقدمه

در سال‌های اخیر، نفوذ بالای سیستم‌های نرم‌افزاری در زندگی انسان منجر به تولید و انباشت حجم انبوهی از اطلاعات به‌صورت دیجیتال شده است. پیش بینی شده که هر ۴۰ ماه مقدار داده‌های تولیدشده در جهان دو برابر می‌شود [۱]. مَنشاء تولید بسیاری از این اطلاعات حسگرها، شبکه‌های اجتماعی و سازمان‌های الکترونیکی است. تعداد کاربران شبکه جهانی اینترنت در سال ۲۰۱۸ از مرز ۴ میلیارد گذشته و به ۴ میلیارد و ۲۱ میلیون رسیده است [۱] با توجه به آمار رسمی جمعیت کره زمین که در این سال ۷ میلیارد و ۵۹۳ میلیون تخمین زده می‌شود، ضریب نفوذ اینترنت در زندگی بَشَر ۵۳ درصد است. رشد سریع سیستم‌های اینترنت پایه عامل اصلی پیدایش جرایم سایبری شامل هَک، فیشینگ[1]، سرقت اطلاعات/هویت و گسترش بدافزارها بوده است [۲]. در سبک زندگی جدید عمده دارایی‌های اشخاص و سازمان‌ها به‌صورت دیجیتال توسط سیستم‌های نرم‌افزاری تولید و ذخیره می‌شوند. ارزشمند بودن این اطلاعات سبب شده که همزمان با گسترش دنیای دیجیتال در زندگی بشر، افراد سود جو و بدخواه نیز بدین عرصه ورود کرده و به‌دنبال اهداف بدخواهانه خود باشند. حملات سایبری سازمان یافته، توسعه و انتقال برنامه‌های مخرب و بدافزارها از جمله این اهداف بدخواهانه هستند که مقابله با آن‌ها مستلزم کسب دانش و توسعه سیستم‌های دفاعی است. اگرچه همواره توسعه سیستم‌های مخرب سریع‌تر از سیستم‌های امنیتی بوده است [۳]. بسیاری از تهدیدات و حملات سایبری توسط بدافزارها انجام می‌شود. عامل رُخداد ۹۰ درصد از خرابی‌ها در سیستم‌های نرم‌افزاری ناشی از وجود و فعالیت بدافزارها است [۴].

ضرورت و اهمیت این پژوهش را می‌توان در رشد روز افزون بدافزارها که حاصل افزایش چشمگیر ضریب نفوذ تجهیزات دیجیتالی در زندگی است [۵] و همچنین پیشرفت و پیچیدگی رفتار آن‌ها دانست. این رشد روز افزون و پیچیدگی بسیار بالای بدافزارها تشخیص و مقابله با آن‌ها را بسیار سخت و مستلزم توسعه روش‌ها و تجهیزات مدرن می‌کند. بنابر گزاش شرکت مَک‌آفی[2] در سال ۲۰۱۳ روزانه بیش از ۱۰۰ هزار بدافزار جدید تولید شده است [۶] یعنی ۶۹ بدافزار در هر دقیقه بنابراین فرکانس تولید بدافزار برای سال مذکور تقریباً ۱ بدافزار بر ثانیه بوده است. در سال ۲۰۱۴، این مقدار به ۳۹۰ هزار بدافزار در هر روز رسیده است [۷] و این به معنای فرکانس تولید ۴/۵ بدافزار بر ثانیه است که حکایت از رشد نزدیک به ۳۰۰ درصدی در طی یک سال داشته است در سال‌های اخیر این رشد همچنان ادامه داشته است. مجموع کل بدافزارهای کشف شده در سال ۲۰۱۶ نسبت به سال ۲۰۱۴ به میزان ۳۶ درصد رشد کرده است [۸]. همچنین گزارش شرکت کَسپراِسکی[3] نشان می‌دهد که تعداد بدافزارهای کشف‌شده توسط محصولاتش در سال ۲۰۱۶ به ۳۲۳ هزار بدافزار در روز رسیده است [۹] که روزانه ۱۳ هزار بدافزار بیشتر از سال ۲۰۱۵ بوده است و در نهایت در سال ۲۰۱۷ تعداد ۶۷۰ میلیون بدافزار توسط آزمایشگاه مَک‌آفی کشف شده است [۱۰]. علت آهنگ بالای تولید بدافزار، ابزارهای خودکار تولید کد، مبهم‌ساز[4] و چندریخت ساز[5] است.

آهنگ رو به رشد بدافزارها در دیگر سیستم‌عامل‌ها از جمله اَندروید، iOS و مَک[6] نیز دیده می‌شود. امروزه اَندروید محبوب‌ترین سیستم‌عامل برای گوشی‌های هوشمند در جهان است [۱۱] به همین دلیل مورد توجه بدافزارها قرار گرفته است [۱۲]. سیستم‌عامل اَندروید هدف ۹۷ درصد از حملات جهان موبایل محور در سال ۲۰۱۳ بوده و در سال ۲۰۱۶ تعداد تروجان‌های کشف‌شده در این سیستم‌عامل به ۷۰۰ هزار نمونه رسیده است [۱۳]. اگر چه بدافزارها در دیگر سیستم‌عامل‌ها نظیر اَندروید رشد قابل توجهی داشته‌اند اما همچنان جهت‌گیری اصلی بدافزارها به سمت سیستم‌عامل ویندوز است به‌گونه‌ای که بیش از ۹۰ درصد کل بدافزارهای فعال در جهان خاص این سیستم‌عامل طراحی شده‌اند [۱۴]. شکل (۱) مقایسه دقیقی از آهنگ رشد بدافزارها در سیستم‌عامل‌های مختلف را در بین سال‌های ۲۰۰۹ تا ۲۰۱۸ نشان می‌دهد. بر این اساس آهنگ رشد بدافزار در سیستم‌عامل ویندوز ۲۴ برابر سیستم‌عامل اَندروید در سال ۲۰۱۸ بوده است.



شکل ۱. آمار رشد بدافزار در سیستم‌عامل‌های مختلف در ۱۰ سال اخیر [۷].

[1] Phishing
[2] McAfee
[3] Kaspersky
[4] Obfuscator
[5] Metamorphic Engine
[6] Mac

علاوه بر اینکه رشد بدافزارها در سال‌های اخیر به شدت افزایش یافته است. خود بدافزارها نیز در حال پیچیده‌تر شدن، هوشمند و هدفمند شدن هستند [۱۵]. بدافزارهای مدرن را از چند جنبه مختلف می‌توان با بدافزارهای سنتی مقایسه کرد [۱۶]. این مقایسه در جدول (۱) نشان داده شده است.

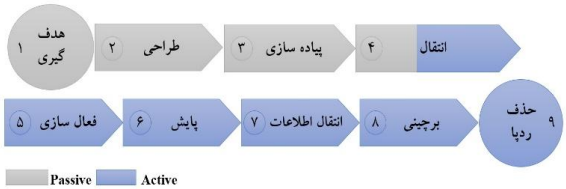
جدول ۱. مقایسه بدافزارهای سنتی و نوین [۱۶]

ردیف	وجه مقایسه	بدافزارهای سنتی	بدافزارهای نوین
۱	هدف حمله	عام منظوره	خاص منظوره
۲	ماهیت بدافزار	فقط بار اول ناشناس	همیشه ناشناس
۳	توانایی مخفی ماندن	بسیار کم	بسیار زیاد
۴	مانایی تهدید	یک‌بار مصرف	ماموریت بلند مدت
۵	محل اقامت	حافظه‌های جانبی	بدون فایل و مقیم پروفایل
۶	سهولت در انتشار	پایین	بسیار بالا

همان‌گونه که در جدول (۱) نشان داده شد رفتار بدافزارهای نوین به شدت مبهم و مخفی‌کارانه است. یکی از پرکاربردترین روش‌های مخفی‌سازی رفتار در بدافزارها تزریق کد و کتابخانه به حافظه اجرایی برنامه‌های دیگر است. بدین ترتیب بدافزار رفتار مخرب خود را با سوء استفاده از امتیاز و امضاء دیگر برنامه‌ها انجام می‌دهد. بدافزار با توزیع کردن دنباله رفتارهای مخرب خود بین فرآیندهای در حال اجرا ابزارهای نظارتی از جمله ضدبدافزارها را فریب می‌دهد. هدف اصلی این مقاله کشف زود هنگام رفتار مخرب تزریق کد و کتابخانه است. این مقاله در پنج قسمت سازمان‌دهی شده است در قسمت دوم روش‌های تشخیص بدافزار و روش‌های مبهم‌سازی بیان می‌شود. قسمت سوم مقاله ضمن تشریح روش تزریق کد و کتابخانه، روش پیشنهادی برای کشف و مقابله را بیان می‌کند و در نهایت قسمت پایانی مقاله، دقت روش پیشنهادی را ارزیابی کرده و ضمن تحلیل نتایج به‌دست‌آمده، آن‌ها را با سایر روش‌ها مقایسه می‌کند.

۲. پیشینه تحقیق

بدافزارها از زمان طراحی تا زمان تکمیل ماموریت که عمر آن‌ها به اتمام می‌رسد فازهای مختلفی را سپری می‌کنند. از دیدگاه معماری نرم‌افزار بدافزار در طول چرخه حیات خود، خواص کیفی[1] (نیازهای غیر وظیفه‌مندی[2]) متناقضی دارد از این رو طراحان باید ضمن توجه به اهمیت ماموریت بدافزار، تعادل بین نیازهای متناقض را حفظ کنند [۱۷]. به‌عنوان نمونه یک بدافزار که قصد سرقت اطلاعات کاربر را داشته باشد مانند جاسوس افزارها باید قابلیت مخفی‌سازی رفتار را در کدهای اجرایی خود تعبیه کند به‌عنوان مثال از ورود به فضای هسته تا حد ممکن پرهیز کند. حال اگر همین بدافزار نیازمند قابلیت خود حفاظتی باشد باید وارد فضای هسته سیستم‌عامل شده و اقدام به نصب درایورهای محافظ نماید که این رفتار با قابلیت مخفی‌سازی کاملاً در تضاد است. به‌طور کلی می‌توان چرخه حیات بدافزار[3] را در ۹ گام خلاصه کرد. این گام‌ها در شکل (۲) نشان داده شده است. اطلاع از این موارد برای مقابله با بدافزار بسیار حایز اهمیت است زیرا در هنگام مواجه با یک بدافزار می‌توانیم تعیین کنیم که بدافزار در کدام فاز از حیاتش است و متناسب با آن تمهیدات لازم برای مقابله را طراحی و اعمال کنیم.



شکل ۲. چرخه حیات بدافزار [۱۷].

در گام اول که هدف‌گیری نامیده می‌شود طراحان به دنبال شناخت سیستم هدف هستند تا بر اساس مشخصات و پیکربندی سیستم هدف بدافزار را طراحی نمایند. به‌عنوان نمونه اگر سیستم هدف از سیستم‌عامل ویندوز استفاده می‌کند شرایط کاملاً متفاوت از حالتی است که هدف از سیستم‌عامل لینوکس بهره می‌برد. همچنین نسخه‌های مختلف یک سیستم‌عامل نیز دارای ماژول‌ها و قالب‌های کاری متفاوت برای برنامه‌نویسی به‌ویژه برنامه‌نویسی سطح پایین هستند لذا شناخت کامل هدف اولین و مهم‌ترین گام در طراحی بدافزار است. در گام دوم و سوم با توجه به پیکربندی سیستم هدف و همچنین امکانات در دسترس برای توسعه بدافزارنویسان اقدام به طراحی و پیاده‌سازی بدافزار می‌نمایند. در این گام موارد متعددی از قبیل نیازهای وظیفه‌مندی و غیر وظیفه‌مندی در نظر گرفته می‌شود و الگوریتم‌های لازم در بدنه بدافزارها تعبیه می‌شود. آزمایش صحت عملکرد نیز در این گام صورت می‌پذیرد. گام چهارم یکی از حساس‌ترین گام‌ها برای بدافزار است زیرا بدافزار از دوره زندگی منفعل[4] به دوره زندگی فعال[5] گذر می‌کند. بدفزار در این گام سعی در یافتن و نفوذ به سیستم هدف دارد. این فرآیند یا بر اساس کدهای سخت‌شده[6] در بدنه بدافزار به‌صورت پیش‌فرض انجام می‌شود یا آنکه توسط مرکز کنترل مدیریت[7] از راه دور هدایت می‌شود. در گام پنجم بدافزار پس از انتقال موفق به سیستم هدف، بنابر فرمان دریافتی از مرکز کنترل مدیریت و یا بر اساس الگوریتم‌های درونی خود در

[1] Quality Attribute
[2] Non Functional Requirements

[3] Malware Lifecycle
[4] Passive
[5] Active
[6] Hardcode
[7] Command and Control (C&C)

زمان یا شرایط خاصی فعال می‌شود. امکان دارد هیچ گونه شرطی برای فعالیت بدافزار وجود نداشته باشد و بدافزار بلافاصله پس از رسیدن به سیستم هدف فعال شود. در گام ششم یا نظارت، بدافزار وظیفه دارد پس از انتقال به سیستم هدف در آنجا گزارشی از محیط پیرامون خود به مرکز کنترل مدیریت ارائه نماید. همچنین گزارش شناسایی و یا عدم شناسایی اهداف نیز در این مرحله به اطلاع مرکز کنترل مدیریت رسانده می‌شود و بدافزار برای دریافت فرمان آماده می‌شود. این نکته بسیار حایز اهمیت است که بدافزارهایی که برای نفوذ به سیستم‌های ایزوله[1] طراحی می‌شوند باید به‌صورت هوشمند طراحی شده تا بتوانند بدون نیاز به ارتباط با مرکز کنترل مدیریت برای کشف و انجام عمل بدخواهانه خود تصمیم‌گیری نمایند. در گام هفتم بدافزار عملیات بدخواهانه خود را انجام داده و نتیجه را به اطلاع مرکز کنترل مدیریت خود اعلام می‌کند با توجه به نوع بدافزار این گام می‌تواند متفاوت باشد به‌عنوان نمونه در جاسوس افزارها تبادل اطلاعات بین بدافزار و مرکز کنترل مدیریت به‌صورت پیوسته برقرار است حال آنکه یک بدافزار تخریبی پس از انجام اقدام مخرب خود با ارسال یک پیام تاییده به مرکز کنترل مدیریت اتمام ماموریت خود را اعلام می‌کند. در گام هشتم پس از اتمام ماموریت بدافزار و یا شرایط اضطرار که از سوی مرکز کنترل و مدیریت اعلام شود بدافزار شروع به حذف و لغو تغییرات خود در سیستم می‌کند این تغییرات می‌تواند بازیابی تنظیمات اولیه رجیستری، حذف کانال‌های پوششی و بستن درگاه‌های بازشده باشد. در گام آخر بدافزار پس از برچینی ردپای خود اقدام به خودکشی[2] می‌کند یعنی اینکه فایل اجرایی خود را از روی دیسک سخت و کدهای خود را از حافظه سیستم حذف می‌نماید.

۲-۱. روش‌های تشخیص بدافزار

در این قسمت پر استنادترین روش‌هایی که در سال‌های اخیر برای تشخیص بدافزارها بر اساس تحلیل رفتار ایستا یا پویا ارائه شده است معرفی و توضیح داده می‌شود.

روش عَلَم و همکاران [۲۰] یک چارچوب برای تشخیص بدافزارهای چند ریخت بر اساس تحلیل گراف جریان کنترلی و فرکانس تغییر Opcode ها ارائه می‌کند. نویسندگان دقت روش خود را بین ۹۴ تا ۹۹/۶ درصد ارزیابی کرده‌اند ولی داده آزمون آن‌ها نامشخص است.

روش وَنگ و همکارن [۲۱] از یک کلاسه‌بند جدید بر اساس الگوریتم ماشین بردار پشتیبان[3] برای حل مساله دقت دسته‌بندی بدافزارهای ناشناخته بر اساس امضاءهای کامل و آموزش دیده شده استفاده می‌کند. مدل آن‌ها قادر است ۶۰ خانواده از بدافزارها را به‌صورت خودکار دسته‌بندی کند. دقت کلی روش پیشنهادی نویسندگان بیش از ۸۵ درصد برای تشخیص بدافزارهای ناشناس تخمین زده شده است.

روش جواهری و همکاران [۲۲] قادر به تشخیص ۵ خانواده از بدافزارها و ۲ خانواده از فایل‌های بی‌خطر است. با داده‌کاوی بروی ۱۵۰۰۰ بدافزار و ۱۳۵۰۰ فایل بی‌خطر خواص ۱۲ گانه از ساختار PE[4] و سرآیند آن همچون نام، تعداد و اندازه قسمت‌ها[5]، میزان بی‌نظمی[6] فایل، نقطه شروع کدهای اجرایی در برنامه و نام کتابخانه‌ها و توابع مورد استخراج شده است که می‌تواند بدافزارها را با دقت ۹۵ درصدی تشخیص داده و دسته‌بندی نماید.

لیو و همکاران [۲۳] یک روش تشخیص بدافزار جدید بر اساس یادگیری ماشین ارائه کرده‌اند. روش نویسندگان با استفاده از الگوی n-gram از Opcode های بدافزارها تصاویر سیاه و سفید ایجاد کرده که این تصاویر برای استخراج ویژگی‌ها به‌منظور خوشه‌بندی بدافزارهای ناشناس بر اساس الگوریتم نزدیک‌ترین همسایه مشترک[7] استفاده می‌شود. نویسندگان از یک مجموعه داده شامل ۲۰۰۰۰ نمونه بدافزار و فایل بی‌خطر برای ارزیابی مدلشان استفاده کرده‌اند. نتایج ارزیابی آن‌ها نشان می‌دهد که بهترین دقت برای دسته‌بندی بدافزارهای ناشناس ۹۸/۹ درصد بوده است همچنین دقت میانگین روش آن‌ها برای تشخیص بدافزارهای مدرن ۸۶/۷ درصد است.

مُهایسِن و همکاران [۲۴] یک روش تشخیص بدافزار با استفاده از تحلیل خودکار رفتار پویا ارائه کرده‌اند. نویسندگان اِمکان تشخیص بدافزارهای ناشناخته را به‌صورت دستی در نظر گرفته‌اند. نویسندگان از چهار منبع سیستمی شامل رجیستری، حافظه، شبکه و فایل‌سیستم برای استخراج و مدل‌سازی رفتار استفاده کرده‌اند. داده‌کاوی نویسندگان در مقیاس ۴۰۰۰ بدافزار انجام شده است که نویسندگان تعمیم آن به ۱۱۵۰۰۰ نمونه را نیز آزمایش کرده‌اند. نویسندگان از یادگیری با سرپرست به دقت ۹۹/۶ درصد برای آماره Recall و ۹۹/۵ درصد برای آماره Precision رسیده‌اند. دقت روش مذکور در یادگیری بدون سرپرست بیش از ۹۸ درصد ارزیابی شده است.

[1] Air Gapped System
[2] Melt
[3] Support Vector Machine (SVM)
[4] Portable Executable
[5] Sections
[6] Entropy
[7] Shared Nearest Neighbor (SSN)

هَنسِن و همکاران [۲۵] با سفارشی‌سازی جعبه‌شن Cuckoo ابزاری برای تحلیل حجم انبوه بدافزارهایی که برای تحلیل ایستا مبهم بودند ارائه کرده‌اند. نویسندگان از الگوریتم RFC[1] برای تشخیص و دسته‌بندی استفاده کرده‌اند. مجموعه داده نویسندگان از ۲۷۰۰۰۰ بدافزار و ۸۳۷ بی‌خطر تشکیل شده است.

ایمران و همکاران [۲۶] از روش تحلیل نمادین[2] برای تشخیص بدافزارها استفاده کرده‌اند. روش آن‌ها یک روش ترکیبی بوده که از فرخوانی‌های سیستمی استخراج‌شده از تحلیل ایستا و پویا به‌عنوان نماد استفاده کرده است. نویسندگان از مدل مارکوف مخفی[3] برای دسته‌بندی بهره برده‌اند.

داس و همکاران [۲۷] یک روش کلی برای تشخیص بدافزارها به‌صورت برخط با استفاده از معماری توسعه یافته سخت‌افزار بواسطه ترکیب پردازنده و FPGA[4] مطرح کرده‌اند. نویسندگان از مدل فرکانس محور برای تهیه ویژگی‌های مورد نیاز داده‌کاوی بدافزارها و فایل‌های بی‌خطر بر اساس الگوی فراخوانی‌های سیستمی استفاده کرده‌اند. نوسندگان ادعا کرده‌اند که مدل آن‌ها برای تشخیص زودهنگام قادر است ۴۶ درصد بدافزارها را در طی ۳۰ درصد ابتدایی اجرا تشخیص دهد و در صورتی که اجرا کامل باشد دقت تشخیص روش پیشنهادی نویسندگان ۹۷ درصد خواهد بود.

۲-۲. مبهم‌سازی رفتار با تزرق کد و ربایش جریان اجرایی

با توجه به استفاده از ابزارهای ضدبدافزار در مقیاس‌های وسیع، یکی از برترین راهکارهای مبهم‌سازی رفتار، توزیع کردن تمام یا بخشی از کدهای بدخواه در برنامه‌های بی‌خطر دیگر و اجرای آن‌ها با نام و نشان دیگر برنامه‌ها است. این روش با سوء استفاده از امضاء و سطح دسترسی برنامه‌های دیگر بسیاری از روش‌های تشخیص بدافزار را فریب می‌دهد و عمده روش‌های تشریح‌شده در قسمت ۲-۱ این مقاله امکان استخراج ویژگی‌های لازم برای تحلیل رفتار را از دست می‌دهند [۲۸]. به‌عنوان مثال یک بدافزار در سطح کاربر می‌تواند با تزریق کردن بخشی از کدهای بدخواه خود در بدنه یک برنامه بی‌خطر با حق دسترسی مدیر سیستم از مجوزهای[5] وی برای اهداف بدخواهانه خود سود ببرد و برنامه ضدبدافزار نمی‌تواند فرآیند رهگیری و تشخیص رفتار بدخواهانه مذکور و منبع آن را به صورت صحیح کشف نماید. این روش به شدت مورد علاقه جاسوس افزارها است که بواسطه آن بتوانند ناشناس ماندن خود را برای زمان طولانی حفظ نمایند [۱۹].

[1] Random Forest Classifier
[2] Symbolic Analysis
[3] Hidden Markov Model (HMM)
[4] Field-Programmable Gate Array
[5] Privileges

حمله خطرناک دیگر تزریق کتابخانه به درون حافظه یک برنامه در زمان اجرا است. هدف اصلی از اینگونه حملات رهگیری، منحرف‌سازی و انسداد فراخوانی‌های سیستمی برای تغییر در رفتار برنامه به منظور ربایش کنترل جریان اجرایی[6] برنامه قربانی است. البته لازم به ذکر است که از روش تزریق کتابخانه برای رهگیری توابع سیستمی در سطح کاربر به منظور تحلیل رفتار بدافزارها در برنامه‌های امنیتی نیز استفاده می‌شود که این کاربرد بدخواهانه نیست لذا باید برای استفاده از این روش توسط برنامه‌های بدخواه و برنامه‌های امنیتی تمایز قائل شد. بدافزاهای بکارگیرنده این روش تحت عنوان آلوده کننده[7] و چسبنده[8] شناخته می‌شوند. مجموع حملات تزریقی شامل تزریق کد/کتابخانه، تزریق پرس‌وجو و تزریق اسکریپت از سال ۲۰۱۳ تا سال ۲۰۱۷ در صدر حملات سایبری جهان بوده است [۱۸].

در حمله تزریق کتابخانه، یک کتابخانه حاوی روتین‌های جعلی از توابع سیستمی توسط یک برنامه تزریق کننده[9] به حافظه برنامه قربانی تزریق می‌شود سپس با استفاده از یک Stub منحرف‌ساز[10] درخواست‌های برنامه قربانی برای استفاده از توابع سیستمی به توابع جعلی بارگذاری‌شده در حافظه فرآیند قربانی منحرف می‌شود. برنامه منحرف‌سازی با ایجاد تغییر در آدرس ذخیره شده در جدول آدرس توابع ورودی[11] فرآیند قلاب‌سازی را تکمیل می‌کند. جدول آدرس توابع ورودی یا IAT جدولی است که در آن نام توابع سیستمی (API) فراخوانی شده توسط برنامه، آدرس آن‌ها در حافظه و نام ماژول مالک توابع ذخیره می‌شود تا برنامه پیونددهنده[12] بتواند در زمان مورد نیاز، تابع سیستمی فراخوانی شده و ماژول مالک را در اختیار برنامه فراخوانی‌کننده قرار دهد. حال بدافزارهای کلاس آلوده‌کننده با علم به این موضوع مبادرت به جایگزین کردن آدرس توابع جعلی خود که از پیش توسط کتابخانه تزریق شوند در حافظه برنامه قربانی بارگذاری شده است می‌نمایند تا برنامه قربانی به‌جای فراخوانی توابع اصلی سیستم‌عامل، توابع جعلی تزریق‌شده را فراخوانی نماید بدین ترتیب بدافزار می‌تواند جریان کنترل اجرایی یک برنامه کاربردی را در اختیار گرفته و آن را مجبور به اجرای کدهای مخرب خود کند.

چالش عمده روش‌های تشخیص بدافزار در مواجهه با این کلاس از بدافزارها، عدم امکان استخراج ویژگی‌های لازم برای تحلیل رفتار است زیرا؛ بدافزار با توزیع کردن کدهای مخرب

[6] Execution Flow Control Hijacking
[7] Infector
[8] Binder
[9] Injector
[10] Redirector
[11] Import Address Table (IAT)
[12] Linker

خود میان چندین فرآیند امکان رصد کردن دنباله رفتارهای مخرب را از دید ناظر امنیتی زمان اجرا سلب می‌کند. همچنین با رمزنگاری کدهای مخرب در زمان ذخیره بروی دیسک سخت امکان هرگونه تحلیل و یا مهندسی معکوس از پویش‌گرهای ایستا نیز گرفته می‌شود [۲۸] با توجه به چالش موجود این مقاله راهکاری دقیق برای کشف زودهنگام و مقابله با رفتار مخرب تزریق کد/کتابخانه پیشنهاد می‌کند.

۳. روش تحقیق

همان‌گونه که در قسمت ۲-۲ ذکر شد حمله تزریق کد و کتابخانه توسط بدافزارها با هدف رهایش کنترل جریان اجرایی قربانی صورت می‌گیرد. رهایش کنترل جریان اجرایی به بدافزار این امکان را می‌دهد که قربانی را وادار به اجرای بخشی از کدهای بدخواه خود کند. بدین ترتیب برنامه‌های رصدگر نمی‌توانند به فایل و فرآیند جاسوس افزار اصلی را به درستی کشف نمایند. راهکار پیشنهادی این مقاله کشف زودهنگام فرآیند تزریق کد یا کتابخانه است تا امکان ردگیری مَنشاء جهت کشف بدافزار وجود داشته باشد. منظور از زودهنگام این است که روش پیشنهادی بتواند قبل از تکمیل شدن فرآیند تزریق، آن را تشخیص دهد زیرا بعد از تزریق دیگر امکان کشف مَنشاء تزریق وجود ندارد و همچنین کنترل جریان اجرایی نیز از دست رفته است و بازگردانی آن مستلزم ختم فرآیند قربانی و راه‌اندازی مجدد سیستم است.

تزریق به حافظه یک برنامه زمانی که در حال اجرا است متفاوت و بسیار سخت‌تر از زمانی است که به‌صورت فایل باینری بروی دیسک سخت ذخیره شده است. بنابراین، تزریق کد و کتابخانه می‌تواند به‌صورت پویا برای یک برنامه در حال اجرا و یا به‌صورت ایستا برای یک برنامه ذخیره‌شده روی دیسک سخت صورت پذیرد که در ادامه تشریح می‌شوند.

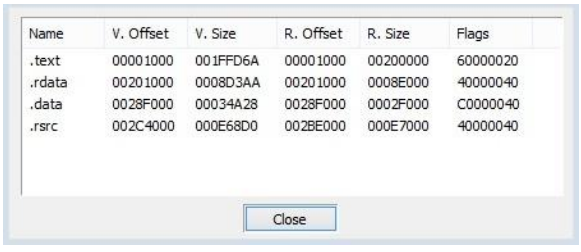
۳-۱. تزریق به فایل باینری

در این روش هدف حمله یک فایل باینری ذخیره‌شده روی دیسک سخت بوده و فرآیند تزریق بدین صورت است که ابتدا یک دستگیره به فایل هدف با استفاده از توابع سیستمی CreateFile و یا OpenFile ایجاد می‌شود در گام بعد آدرس نقطه شروع برنامه محاسبه می‌شود. نقطه شروع برنامه یا EP[1] محلی است که کدهای اجرایی برنامه از آن نقطه شروع به اجرا می‌کنند. آدرس این نقطه بر اساس معادله (۱) محاسبه می‌شود در ادامه بعد از به‌دست آوردن نقطه ورود فایل اجرایی، یک قسمت جدید به فایل PE اضافه می‌شود و با استفاده از تابع سیستمی WriteFile کدهای مخرب در قسمت جدید که اخیراً ایجاد شده نوشته می‌شوند سپس مقدار OEP[2] بروی آدرس آغازین قسمت جدید تنظیم می‌شود. علت این امر آن است که در ابتدای هر بار اجرای برنامه کدهای بدخواه اجرا شوند و مهاجم به اهداف بدخواهانه خود دست پیدا کند.

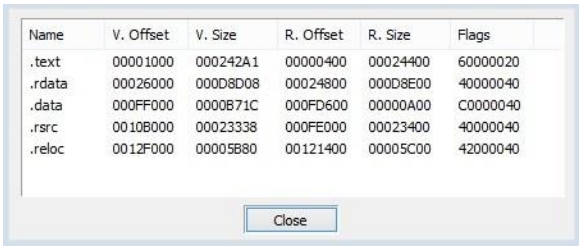
$$EP = ImageBase + C \qquad (۱)$$

در این معادله نقطه شروع برنامه از حاصل جمع محل شروع کد در حافظه یا ImageBase با یک مقدار ثابت C به‌دست آمده است [۲۲]. مقدار ImageBase تا نسخه NT 6.0 از سیستم‌عامل ویندوز همواره ثابت و برابر ۴۰۰۰۰۰ بود لذا محاسبه EP بسیار ساده بود. برای مقابله با حملات سرریز بافر و سرریز Heap فناوری ASLR[3] توسط مایکروسافت در نسخه‌های بعدی ویندوز مورد استفاده قرار گرفت. این فناوری ضمن بارگذاری تصادفی کتابخانه‌های مورد استفاده یک برنامه، ناحیه‌های تخصیص‌یافته به فضاهای کُپه و پُشته را نیز به‌صورت تصادفی در هر بار اجرا تغییر می‌دهد لذا آدرس ImageBase نیز در هر بار اجرا تغییر می‌کند. این فناوری به‌صورت گسترده توسط دیگر سیستم‌عامل‌ها از جمله لینوکس با هسته 2.6+، آندروید از نسخه 4.0، سُلاریس از نسخه 11.1 و iOS از نسخه 4.3 مورد استفاده قرار گرفته است. اگر چه به‌کارگیری فناوری ASLR باعث کاهش قابل توجه حملات تزریق کد، شِل کدنویسی و سرریز بافر و کُپه شد ولی کار محاسبه EP را سخت کرده و باعث افزایش آنتروپی فایل باینری می‌شود.

در شکل‌های (۴-۳) قسمت‌ها و پرچم‌های ساختار PE یک برنامه کاربردی که هدف حمله بدافزار جهت تزریق کد مخرب قرار گرفته است را به‌ترتیب قبل و بعد از فرآیند تزریق نشان می‌دهند. بدافزار با ایجاد یک قسمت جدید و تغییر در آدرس‌های سایر قسمت‌ها کدهای مخرب خود را در زمان آغاز اجرای برنامه قربانی اجرا می‌کند.



شکل ۳. ساختار PE برنامه بی‌خطر قبل از تزریق کد توسط بدافزار



شکل ۴. ساختار PE همان برنامه بعد از تزریق کد توسط بدافزار

[1] Entry Point
[2] Original Entry Point
[3] Address Space Layout Randomization

۳-۲. تزریق به فرآیند در حال اجرا

تزریق به فرآیند در حال اجرا به‌دلیل اعمال سیاست‌های محدودیتی سیستم‌عامل از جمله UAC[1] و KPP[2] بروی حافظه اصلی بسیار سخت‌تر از تزریق به فایل باینری است. تزریق به فرآیند در حال اجرا سبب می‌شود که کدهای تزریقی بدافزار به‌صورت بلادرنگ بعد از فرآیند تزریق اجرا شوند در صورتی که در حالت قبل کدهای تزریقی تا زمان بارگذاری‌شده فایل قربانی در حافظه اصلی قابل اجرا نیستند. یکی از روش‌های قدیمی برای تزریق کتابخانه به یک فرآیند ثبت، نام و آدرس آن در کلید رجیستری AppInitDlls است [۲۹]. سیستم‌عامل به هنگام بارگذاری، تمام کتابخانه‌های لیست‌شده در این کلید رجیستری را به‌صورت خودکار در فرآیندهای هدف بارگذاری می‌کند با توجه به سوء استفاده از این کلید رجیستری برای انجام حمله تزریق کتابخانه، شرکت سازنده یعنی مایکروسافت بارگذاری کتابخانه بواسطه کلید رجیستری مذکور را منوط به داشتن امضاء معتبر از سوی خویش کرد بدین ترتیب برنامه‌نویسان بدخواه در استفاده از این روش دچار محدودیت شدند لذا به راهکارهای دیگر ایجاد شد که ادامه تشریح می‌شود.

با انجام داده‌کاوی در حجم انبوه از بدافزارهایی که توانایی تزریق کد یا کتابخانه دارند روش کار آن‌ها مشخص شد. این بدافزارها در ابتدا یک دستگیره به فرآیند قربانی که در حال اجرا است، ایجاد می‌کنند این دستگیره با استفاده از توابع سیستمی CreateProcess و یا OpenProcess ایجاد می‌شود بعد از ایجاد دستگیره با استفاده از تابع سیستمی ReadProcessMemory حافظه اختصاص یافته به فرآیند در حال اجرا را مورد دسترسی قرار داده و پس از آن با استفاده از تابع سیستمی VirtualAlloceEX مقدار مشخصی فضای آزاد به حجم فضای اختصاص داده شده به فرآیند قربانی اضافه می‌کنند. این فضا معمولاً به اندازه حجم کد و یا کتابخانه تزریق‌شونده است. در گام بعد بدافزار با استفاده از تابع سیستمی WriteProcessMemory کدهای مخرب خود را در فضای آدرس افزوده شده به فرآیند می‌نویسد. دسترسی به این کدها می‌تواند نیت بدافزار را اِفشا نماید.

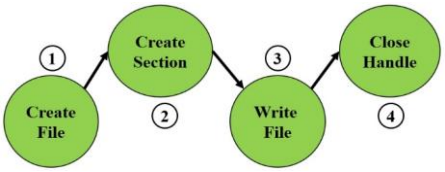
مراحل ذکر شده در قسمت ۳-۲ مربوط به تزریق کد به یک فرآیند در حال اجرا است در صورتی که هدف، تزریق کتابخانه به آن فرآیند باشد کدهای تزریق‌شده کتابخانه جعلی را بارگذاری می‌کنند. روش کار بدین صورت است که در ابتدا معادل اسمبلی تابع سیستمی SuspendThread برای متوقف‌سازی نخ اصلی سپس معادل اسمبلی تابع LoadLibrary به همراه آدرس کتابخانه مورد نظر برای تزریق و در ادامه معادل اسمبلی تابع سیستمی CreateRemoteThread برای ایجاد نخ راه دور در حافظه فرآیند قربانی به‌عنوان کدهای تزریق‌شونده به حافظه فرآیند قربانی با روش تزریق کد، تزریق می‌شوند. همچنین کدهای تزریق‌شونده بایستی کار تنظیم پرچم EIP و ثَبات PC بروری آدرس کدهای تزریق‌شده به‌عنوان نقطه اجرایی برنامه را نیز انجام دهند. در ادامه از تابع سیستمی WaitForSignalObject برای آگاهی از زمان کامل شدن بارگذاری کتابخانه و در پایان از تابع ResumeThread برای ادامه اجرای فرآیند که هم اکنون در حالت تعلیق[3] است استفاده می‌شود. با توجه به امکان سوءاستفاده از تابع سیستمی CreateRemoteThread به‌منظور تزریق کد بدخواه به حافظه دیگر فرآیندها، شرکت سازنده یعنی مایکروسافت محدودیت‌هایی برای استفاده از این تابع در نسخه‌های جدید سیستم‌عامل ویندوز (از ویندوز ویستا به بعد) در نظر گرفته است که مطابق آن دیگر امکان فراخونی این تابع برای ایجاد نخ راه دور میان دو برنامه با سطح دسترسی متفاوت، به‌عنوان نمونه یکی کاربر ساده و دیگری مدیر سیستم و یا فرآیندهایی با دو مالک متفاوت وجود ندارد. اگرچه محدودیت‌های وضع شده در جلوگیری از سوءاستفاده از تابع سیستمی مربوطه تاثیر گذار بود ولیکن همچنان روش‌هایی برای ایجاد نخ راه دور وجود دارد. برنامه‌نویسان بدخواهی که اصرار به توسعه بدافزارشان با سطح دسترسی کاربر برای فریب ابزارهای ضدبدافزار دارند به‌جای استفاده از تابع CreateRemoteThread برای اعمال تغییرات در حافظه فرآیند قربانی از توابع سیستمی GetProcessContext برای دریافت محتوای تنظیمات جاری یک فرآیند استفاده کرده سپس تغییرات مورد نظر خود را با استفاده از دستورات اسمبلی در Context به‌دست آمده اعمال، آنگاه با استفاده از تابع سیستمی SetThreadContext آن‌را به نخ هدف در فرآیند قربانی اعمال می‌کنند. این راهکار اگر چه نیازی به استفاده از تابع سیستمی CreateRemoteThread ندارد ولی سختی پیاده‌سازی را به همراه دارد [۲۹]. لازم به ذکر است که این محدودیت متوجه بدافزارهای سطح هسته نخواهد بود.

۳-۳. کشف حملات تزریق به صورت ایستا

در روش پیشنهادی حملات تزریق کد و کتابخانه در سه لایه تشخیصی کشف می‌شوند. لایه اول با جستجوی زیردنباله‌های یکتا به‌وسیله n-gram، لایه‌های دوم با استخراج فراخوانی‌ها از سرآیند ساختار PE و درنهایت لایه سوم با استخراج فراخوانی‌ها در زمان اجرا (تحلیل پویا) حملات مذکور را کشف می‌نمایند. در لایه اول کشف حمله تزریق کد با اثبات وجود اِستاب‌های

[1] User Access Control
[2] Kernel Path Protection
[3] Suspend

تزریق‌کننده در بدنه فایل اجرایی‌های صورت می‌پذیرد. پرکاربردترین این استاب‌ها EasyHook و Hook Tool SDK برای زبان‌های برنامه‌نویسی دات‌نت، Marshal SDK برای زبان دِلفی و API Hijack برای زبان C/C++ است. تشخیص با جستجوی زیر دنباله‌هایی یکتا از استاب‌های فوق‌الذکر با استفاده از n-gram از بایت‌های ابتدایی و انتهایی صورت می‌پذیرد. مشکل موجود در تشخیص توسط این روش، وجود بدافزارهای بسته‌بندی و محافظت شده[1] است. این گونه بدافزارها که با استفاده از ابزارهای بسته‌بندی‌کننده و مبهم‌ساز بدنه برنامه‌های تحت حفاظت خود را به‌گونه‌ای تغییر می‌دهند که دیگر امکان تهیه امضاء و تحلیل آن وجود نداشته باشد است [۲۸]. لذا لایه دوم تشخیص در روش پیشنهادی یعنی تحلیل ساختار فایل PE برای کشف رفتار بدخواهانه متناظر با حملات تزریق کد وارد عمل می‌شود. برنامه پویش‌گر در این لایه از روش پیشنهادی به دنبال نام توابع متناظر با رفتار بدخواهانه تزریق کد بوده تا با استفاده از آن اقدام به کشف رفتار بدخواهانه نماید. روش پیشنهادی جدول IAT برنامه هدف را به دقت مورد بررسی قرار داده تا نام توابع ذکر شده برای تزریق جستجو و استخراج شود. در روش پیشنهادی توابعی که نمایان‌گر رفتار تزریق کد در فایل باینری برنامه‌های دیگر هستند به‌صورت زنجیره‌ای با چهار گام مطابق شکل (۵) مدل‌سازی می‌شود. روش پیشنهادی بر اساس قوانین تعریف شده، در صورت مشاهده این زنجیره در یک برنامه آن را در به‌عنوان رفتار مخرب و بدخواهانه تزریق کد تشخیص می‌دهد.



شکل ۵. زنجیره پیشنهادی برای تشخیص رفتار تزریق کد در فایل باینری به صورت ایستا

لازم به ذکر است در گذشته رفتار تزریق در یک فایل به‌صورت بالفطره بدخواهانه نبوده است ولی به‌دلیل استفاده مُکَرَر بدافزارها به‌ویژه جاسوس‌افزارها از این روش، امروزه به‌عنوان یک رفتار مخرب تشخیص داده می‌شود.

به‌منظور افزایش دقت تشخیص بدخواهانه بودن رفتار تزریق کد در فراخوانی اولین گام از زنجیره یعنی تابع سیستمی CreateFile یا OpenFile بررسی می‌شود که پارامتر اول آن یعنی ساختار LPCTSTR lpFileName برابر با آدرس فایلی باشد که مالک آن فرآیند جاری (فراخونی‌کننده) نباشد یعنی فرآیند تزریق در فایل برنامه دیگری باشد تا به‌عنوان رفتار بدخواهانه در نظر گرفته شود در غیر این‌صورت فرآیند تزریق برای فایل‌هایی که خود فرآیند تزریق‌کننده مالک آن‌ها است بدخواهانه در نظر گرفته نمی‌شود. به‌منظور افزایش دقت روش پیشنهادی و نگه‌داشتن آهنگ کاذب‌مثبت[2] در حد قابل قبول، میزان بدخیمی فایل تحت نظارت با استفاده از معادله (۲) سنجش می‌شود [۲۲].

$$\text{Suspicious Rate} = \frac{\sum W\times P_i + \sum W\times N_j}{\sum Wi + Wj} \qquad (۲)$$

در این معادله میزان بدخیمی یک فایل با ساختار PE را نشان می‌دهد. N ویژگی‌های منفی و P ویژگی‌های مثبت در ساختار PE است. W نشان‌دهنده ضریب تاثیر هر یک ویژگی‌ها است. ویژگی‌ها شامل اندازه، نام، تعداد و خواص قسمت‌ها، اندازه خام و مجازی، میزان بی‌نظمی و نقطه شروع کدهای اجرایی است که با داده‌کاوی در مقیاس وسیع از بدافزارها و برنامه‌های بی‌خطر استخراج شده است. این مقدار هرچقدر به یک نزدیک‌تر باشد نشان‌دهنده تشابه ساختار فایل با بدافزارها و هرچقدر به صفر نزدیک‌تر باشد نشان دهنده تشابه با بی‌خطرها است.

نقطه ضعف و آسیب‌پذیری روش پویش ایستا در مواجهه با کدهای محافظت‌شده توسط ابزارهای بسته‌بندی و محافظ[3] است. این ابزارها با به‌کارگیری روش‌هایی نظیر رمزنگاری، تخلیه یا تخریب IAT امکان اعمال هرگونه تحلیل به‌صورت ایستا را از تحلیل‌گر سلب می‌کنند. با توجه به چالش‌های موجود، روش پیشنهادی این مقاله راهکار اصلی را در تشخیص بر اساس شنود فراخوانی‌ها در زمان اجرا پیشنهاد می‌کند علت این امر علاوه چالش‌های ذکر شده، محدودیت تحلیل ایستا در پویش فرآیندهای در حال اجرا و همچنین تعیین ترتیب فراخوانی‌ها است. لایه سوم که مبتنی بر تحلیل رفتار پویا است برای مقابله با چالش‌های موجود تعریف می‌شود.

۳-۴. کشف حملات تزریق به صورت پویا

لایه سوم از روش پیشنهادی به کشف حمله تزریق کد/کتابخانه به‌صورت پویا اختصاص دارد. یکی از مزایای روش پیشنهادی در این است که می‌تواند حملات تزریق کتابخانه را قبل از تکمیل شدن کشف نماید لذا امکان خنثی‌سازی حمله و ممانعت از تکمیل آن در روش پیشنهادی وجود دارد. همچنین روش‌هایی که بعد از تزریق و بارگذاری شدن کتابخانه داخل حافظه اختصاص یافته به فرآیند قربانی حمله را کشف می‌کنند مستلزم هزینه دورریز برای تخلیه کتابخانه تزریق‌شده از حافظه فرآیند قربانی هستند که روش پیشنهادی به‌دلیل کشف زودهنگام (قبل از تکمیل حمله) از این هزینه مستثنی است.

[1] Packed and Protected Malwares

[2] False Positive

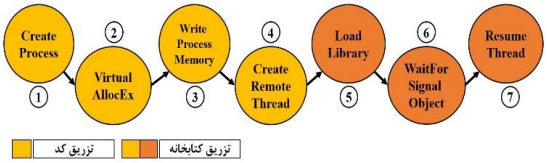
[3] Packer and Protector

کشف زودهنگام حمله موجب حفظ و نگه‌داشت جریان اجرایی برنامه هدف حمله می‌شود. در صورت تکمیل شدن فرآیند حمله و بارگذاری کتابخانه به‌دلیل قلاب‌شدن توابع، احتمال از دست رفتن کنترل اجرایی برنامه با تخریب توابع سیستمی حساس مانند TerminateProcess توسط بدافزار وجود دارد در این حالت ختم فرآیند امکان پذیر نیست [۳۰] لذا کشف زودهنگام و قبل از تکمیل شدن فرآیند تزریق از نوآوری‌های روش این مقاله محسوب می‌شود. روش‌های پیشین برای تشخیص اینگونه حملات امکان کشف زود هنگام را نداشته و علاوه بر آن از امکان مقابله و ایجاد مصونیت در مقابل این گونه حملات نیز محروم بودند. این دو امر مهم با توسعه مدل پیشگو در روش پیشنهادی به دقت پیاده‌سازی شده است. از جمله روش‌های پیشین شمارش کتابخانه‌های موجود در فضایی تخصیص داده شده به یک برنامه و مقایسه آن با تعداد کتابخانه‌های مورد نیاز که از قبل توسط کامپایلر در داده‌های موجود در سرآیند فایل PE قرار داده شده است. در این حالات اگر تعداد کتابخانه‌های بارگذاری‌شده در حافظه فرآیند بیشتر از تعداد مورد نیاز باشد آنگاه می‌توان نتیجه گرفت که یک حمله تزریق کد به برنامه صورت پذیرفته است. با استخراج نام کتابخانه‌های بارگذاری‌شده در حافظه فرآیند و مقایسه آن‌ها با نام کتابخانه‌های مورد نیاز که از قبل در جدول IAT موجود است می‌توان کتابخانه تزریق‌شده را یافت و آن‌را از حافظه فرآیند تخلیه کرد. مشکل این روش در بالا بودن آهنگ کاذب مثبت است همچنین کشف بعد از کامل شدن حمله و تزریق کتابخانه صورت می‌گیرد لذا هیچ‌گونه امکانی برای مقابله و توقف حمله وجود ندارد و سربار اضافی برای تخلیه کتابخانه تزریق شده و احتمال از دست رفتن کنترل برنامه قربانی نیز وجود دارد.

روش پیشنهادی می‌تواند رفتار تزریق کد و کتابخانه را در لایه سوم و به‌صورت پویا و زودهنگام کشف نماید. در روش پیشنهادی رفتار بدخواهانه تزریق کد و کتابخانه با استفاده از ترتیب فراخوانی‌های سیستمی و فرکانس تَکَرُر آن‌ها مدل شده است. برای مدل‌سازی این رفتار از تابع رگرسیون خطی[1] استفاده شده است. در مدل مذکور رفتار بدخواهانه M که نمایانگر تزریق کد است به‌صورت معادله (۳) تعریف شده است [۱۶].

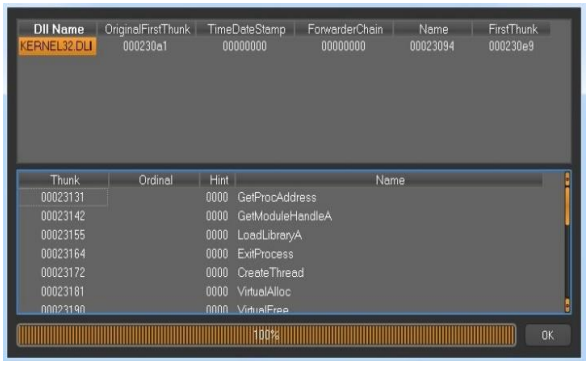
$$M = A \times X_A + B \times X_B + C \times X_C + \cdots \qquad (۳)$$

در این معادله A, B, C فراخوانی‌های سیستی و X_A , X_B , X_C فرکانس تَکَرُر آن‌ها هستند. M زنجیره فراخوانی براساس تَواتُر زمان فراخوانی است. برای استفاده از معادله (۳) در روش پیشنهادی، از توابع سیستمی ذکر شده در شکل (۶) استفاده شده و رفتار بدخواهانه تزریق کد یا کتابخانه مدل می‌شود. بنابراین، در روش پیشنهادی زنجیره‌ای مطابق با شکل (۶) تشکیل می‌شود. در صورتی که یک فرآیند در فراخوانی‌های سیستمی‌اَش چنین زنجیره‌ای داشته باشد روش پیشنهادی به‌صورت بلادرنگ آن‌را به‌عنوان رفتار بدخواهانه تزریق کد یا کتابخانه تشخیص می‌دهد این رفتار نمایان‌گر فعالیت یک بدافزار از کلاس آلوده‌کننده در سیستم است.



شکل ۶. زنجیره پیشنهادی برای تشخیص رفتار تزریق کد و کتابخانه در فرآیند به صورت پویا

همانگونه که در شکل (۶) مشخص است زنجیره تزریق در روش پیشنهادی از ۷ گام تشکیل شده است که نمایانگر رفتار مخرب تزریق کتابخانه است. این در حالی است که ۴ گام ابتدایی آن به تنهایی نشان‌دهنده رفتار بدخواهانه تزریق کد است همچنین لازم به ذکر است با توجه به داده‌کاوی انجام‌شده در حجم انبوه بدافزار گردآوری شده از مراجع [۳۳-۳۱] بعضاً در برخی از بدافزارها در بین گام‌های ۲ و ۳ از زنجیره شکل (۶) تابع سیستمی ReadProcessMemoy و بین گام‌های ۴ و ۵ تابع سیستمی SuspendThread مشاهده شده است. زنجیره پیشنهادی همواره زیر مجموعه‌ای از زنجیره‌های بزرگتر (با ۲ گام ذکر شده) است لذا همیشه می‌تواند رفتار معادل با تزریق را تشخیص داده و سرعت تشخیص آن نیز بیشتر است. همچنین مشاهده شد که برخی از بدافزارها برای ایجاد دستگیره به یک فرآیند در حال اجرا از تابع سیستمی OpenProcess به‌جای CreateProcess استفاده می‌کنند که در روش پیشنهادی این دو تابع معادل در نظر گرفته شده‌اند. در ادامه و در شکل (۷) جدول IAT مربوط به یک بدافزار با توانایی تزریق کد نشان داده شده است. این بدافزار در لایه تشخیصی دوم از روش پیشنهادی کشف شده است.



شکل ۷. توابع موجود در جدول IAT یک جاسوس افزار با توانایی تزریق کتابخانه که توسط روش پیشنهادی تشخیص داده شده

[1] Linear Regression

همان‌گونه که ذکر شد تشخیص حمله تزریق کتابخانه در صورتی که زودهنگام تشخیص داده شود ارزشمند است. برای تحقق این امر در روش پیشنهادی از الگوریتم اَپریوری[1] به‌منظور یادگیری مبتنی بر قواعده انجمی[2] استفاده شده است شبه کد این الگوریتم به صورت زیر است [۳۴]:

$$
\begin{aligned}
&\text{Apriori}(T,\epsilon)\\
&\quad L_1 \leftarrow \{\text{large 1-itemsets}\}\\
&\quad k \leftarrow 2\\
&\quad \text{while } L_{k-1} \neq \emptyset\\
&\qquad C_k \leftarrow \{a \cup \{b\} \mid a \in L_{k-1} \wedge b \notin a\} - \{c \mid \{s \mid s \subseteq c \wedge |s| = k-1\} \nsubseteq L_{k-1}\}\\
&\qquad \text{for transactions } t \in T\\
&\qquad\quad D_t \leftarrow \{c \mid c \in C_k \wedge c \subseteq t\}\\
&\qquad\quad \text{for candidates } c \in D_t\\
&\qquad\qquad count[c] \leftarrow count[c] + 1\\
&\qquad L_k \leftarrow \{c \mid c \in C_k \wedge\ count[c] \geq \epsilon\}\\
&\qquad k \leftarrow k+1\\
&\quad \text{return } \bigcup_k L_k
\end{aligned}
$$

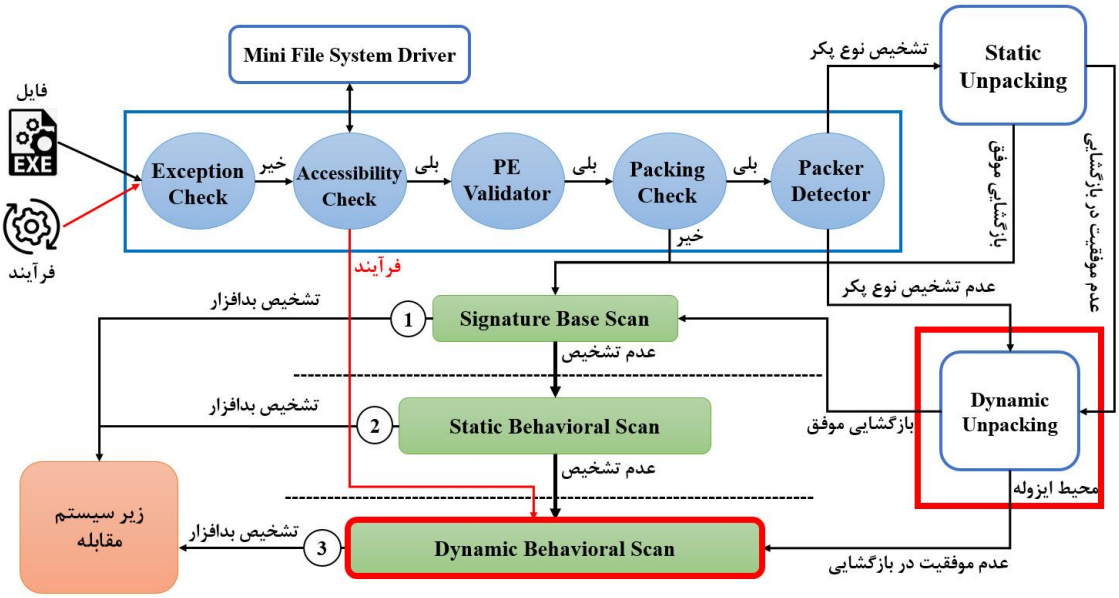
با محاسبه میزان Confidence و Lift برای زنجیره فراخوانی‌های شکل (۶) امکان پیشگویی گام‌های ششم و هفتم زنجیره محقق شده است. بدین ترتیب روش پیشنهادی قبل از تکمیل حمله و تزریق کامل کتابخانه آن را در گام پنجم تشخیص می‌دهد. معیارهای Confidence و Lift بر اساس معادله (۴) محاسبه می‌شوند [۳۴]:

$$\text{Conf}(A \rightarrow B) = \frac{\text{Sup}(A \cup B)}{\text{Sup}(A)} \qquad (۴)$$

$$\text{Lift}(A \rightarrow B) = \frac{\text{Conf}(A \rightarrow B)}{\text{Sup}(B)}$$

در این معادله A, B شاخص فراخوانی‌های سیستمی مطابق با زنجیره شکل (۶) هستند. معیار Confidence نشان‌دهنده میزان وابستگی بین فراخوانی‌ها است در روش پیشنهادی احتمال فراخوانی‌شدن گام‌های ۵ تا ۷ به شرط انجام ۴ گام اول را نشان می‌دهد. معیار Support نشان‌دهنده تعداد فراخوانی‌هایی است که هر دو تابع سیستمی A و B با یکدیگر در فراخوانی حضور دارند نسبت به کل فراخوانی‌ها است. معیار Lift نیز نشان‌دهنده میزان استقلال بین فراخوانی‌ها است. لازم به ذکر است که داده‌کاوی و مدل‌سازی روش پیشنهادی این مقاله در محیط نرم‌افزار Weka نسخه ۲,۷,۹ انجام شده است.

همانگونه که اشاره شد روش پیشنهادی فرآیند تشخیص را در سه لایه متفاوت و در قالب یک برنامه پویش‌گر پیاده‌سازی‌شده به زبان‌های C#.net و C++ انجام می‌دهد. معماری برنامه پویش‌گر و جریان کار[3] آن در شکل (۸) نشان داده شده است.



شکل ۸. معماری سامانه پویش‌گر روش پیشنهادی

سامانه پویش‌گر طراحی‌شده امکان تعبیه‌شدن در یک برنامه ضدبدافزار یا ضد جاسوس‌افزار را دارد. در این سامانه آدرس یک فایل باینری یا شماره یک فرآیند در حال اجرا به‌عنوان ورودی دریافت شده و به ماژول Exception Check که اولین ماژول در سامانه بوده و وظیفه آن کنترل استثنائات است، تحویل می‌شود. این ماژول وظیفه خود را با بررسی وجود هَش فایل ورودی در لیست استثنائات آغاز می‌کند در صورتی که فایل ورودی جزء استثنائات باشد از ادامه پویش کنار گذاشته شده و فایل جدید از ورودی دریافت می‌شود در غیر این صورت فایل

[1] Apriori
[2] Association Rules
[3] Workflow

مذکور به دومین ماژول یعنی Accessibility Check که ماژول کنترل دسترس‌پذیری است داده می‌شود. هدف از تعریف لیست استثنائات جلوگیری از پویش فایل‌های تکراری است. وظیفه ماژول دوم بررسی امکان دسترسی به فایل یا فرآیند ورودی است در صورتی که فایل دسترس‌پذیر نباشد مسیر آن به مینی درایور فایل سیستم سامانه برای خواندن مستقیم از روی دیسک سخت داده می‌شود علت این امر در استقلال از سیستم‌عامل و محدودیت‌های اعمال‌شده توسط UAC و KPP است [۲۹] در ادامه فایل به ماژول PE Validator برای تصدیق ساختار PE تحویل می‌شود، سامانه پویش‌گر در حال حاضر امکان پویش فایل با ساختار PE32 و PE32+ را در قالب‌های exe، dll و sys را دارد. این ماژول معتبر بودن ساختارهای مذکور را کنترل کرده و فایل‌های غیر PE را کنار می‌گذارد.

فایل‌ها با ساختار PE32 و PE32+ معتبر به ماژول Packing Check تحویل می‌شوند وظیفه این ماژول بررسی بسته‌بندی یا محافظت شده بودن فایل است در صورتی که فایل ورودی بسته‌بندی نباشد برای پویش به لایه اول یعنی Signature Base Scan برای پویش مبتنی‌بر امضا تحویل می‌شود در غیر اینصورت به ماژول Packer Detector برای تعیین نوع ابزار بسته‌بندی یا محافظ منتقل می‌شود. این ماژول درصورتی که بتواند نوع ابزار بسته‌بندی کننده را تشخیص دهد، فایل را برای بازگشایی ایستا به ماژول Static Unpacking می‌دهد این ماژول با در اختیار داشتن الگوریتم بازگشایی می‌تواند فایل مبهم‌شده را رفع ابهام کرده و به لایه اول پویش منتقل کند در صورتی که نوع ابزار بسته‌بندی‌کننده تشخیص داده نشود و یا فرآیند بازگشایی ایستا دچار مشکل شود فایل ورودی به ماژول Dynamic Unpacking داده می‌شود. این ماژول فایل ورودی را در محیط ایزوله و تحت نظارت اجرا کرده و از محتویات حافظه آن در زمان مناسب بر اساس روش [۲۸] روبرداری و پس از تعمیر خرابی‌های احتمالی، اطلاعات را در قالب یک فایل جدید بازنویسی می‌کند. فایل بازنویسی‌شده به لایه اول پویش منتقل می‌شود. در این لایه، ماژول تشخیص مبتنی بر امضاء فایل را برای یافتن Stub های بدخواه با استفاده از n-gram پویش می‌کند و در صورتی که امضاء آن به‌عنوان رفتار مخرب در پایگاه داده موجود باشد آن را بدخواه تشخیص می‌دهد در غیر این صورت فایل به لایه پایین‌تر (لایه دوم) یعنی Static Behavioral Scan برای تشخیص مبتنی‌بر تحلیل رفتار ایستا تحویل می‌شود. این تشخیص فارغ از اجرا و فقط با بررسی خواص ساختار PE صورت می‌پذیرد در این لایه با بررسی سرآیند ساختار PE، آهنگ بدخیمی مطابق معادله (۲) مشخص و نام فراخوانی‌های برنامه از جدول IAT استخراج می‌شود و در صورت یافتن فراخوانی‌ها مخرب، کار در این مرحله تمام می‌شود در غیر اینصورت به لایه سوم (لایه آخر) یعنی Dynamic Behavioral Scan برای تشخیص مبتنی بر تحلیل رفتار پویا می‌رویم در این لایه فایل ورودی در محیط ایزوله تحت نظارت و کنترل اجرا شده سپس امکانات رصدگر برنامه اقدام به ضبط فراخوانی‌های سیستمی به‌منظور کشف زنجیره‌های مخرب می‌کنند. اگر پویش‌گر پیشنهادی هیچ زنجیره مخربی را کشف نکند سامانه فایل را بی‌خطر تشخیص می‌دهد و در صورتی که هر کدام از لایه‌ها رفتار مخربی را تشخیص دهند بلافاصله فایل یا فرآیند مذکور به زیر سیستم مقابله برای خنثی‌سازی معرفی می‌شود. لازم به ذکر است که در صورتی که ورودی سامانه یک فرآیند باشد پس از کنترل دسترس‌پذیری مستقیماً به لایه سوم معرفی می‌شود. امکان شنود فراخوانی‌ها بر اساس نصب قلاب‌های شنودگر IRP مطابق با روش [۲۸] محقق می‌شود. علت استفاده از قلاب‌های مبتنی بر IRP[1]، عمق تحلیل بالا و شفافیت زیاد این نوع از قلاب‌ها است که امکان تحلیل بدافزارهای فعال در فضای کاربر و هسته را فراهم می‌کند.

۳-۵. خنثی‌سازی حملات تزریق

این احتمال وجود دارد که برنامه پویش‌گر پیشنهادی هدف حمله بدافزار به‌منظور تزریق کد یا کتابخانه جهت سرقت اطلاعات یا ربایش کنترل اجرایی برنامه، قرار گیرد. از این‌رو، سامانه پیشنهای برای عملکرد صحیح در محیط آلوده باید بتواند با بدافزارهای فعال مقابله کرده و از خود در مقابل حملات تزریقی دفاع کند در این قسمت نحوه محافظت از یک فرآیند در حال اجرا و فایل‌های سامانه پیشنهادی توضیح داده می‌شود. روش این مقاله قادر است وقوع این حملات را به‌صورت زودهنگام و قبل از کامل شدن تشخیص دهد حال برای جلوگیری از تکمیل حمله با محدودسازی برخی از فراخوانی‌های سیستمی از شکل‌گیری زنجیره‌های تزریق که در قسمت ۳-۴ توضیح داده شد، جلوگیری می‌شود. قلاب نصب‌شده بروی توابع زنجیره شکل (۶) به غیر از تابع CreateRemoteThread همگی از نوع Post Callback هستند. قلاب تابع مذکور از نوع Pre Callback است در این نوع قلاب، تابع Callback پیش از انجام فراخوانی تابع اصلی صدا زده می‌شود و این فرصت به پویش‌گر داده می‌شود تا در صورت تشخیص زود هنگام حمله از ادامه فراخوانی‌ها جلوگیری کند. این امر با مسدودسازی تابع CreateRemoteThread از جدول SSDT[2] در صورتی که پارامتر اول یعنی HANDLE hProcess دستگیره‌ای به فرآیند برنامه تحت حفاظت باشد صورت می‌پذیرد. در این حالت ۴ گام اول از فراخوانی‌های زنجیره ۷

[1] I/O Request Packet
[2] System Service Descriptor Table

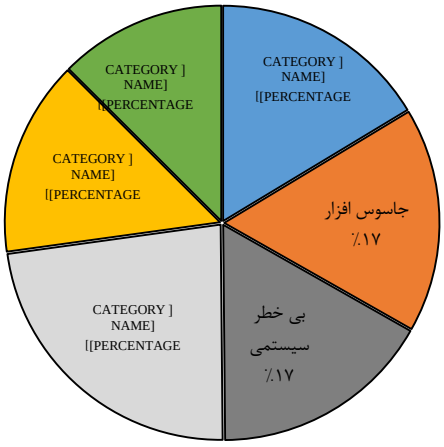
مرحله‌ای شکل (۶) انجام می‌شود زیرا نمی‌توان تا رسیدن به گام پنجم ماهیت رفتار مخرب تزریق کد/کتابخانه را به‌طور دقیق پیشگویی و از انجام فراخوانی‌های بعدی جلوگیری کرد ولی قلاب نصب‌شده بر تابع CreateRemoteThread از انجام گام پنجم فراخوانی با دورریز درخواست وارده جلوگیری می‌کند. بدین ترتیب مهاجم هیچ‌گاه امکان ایجاد نخ راه دور در فرآیند تحت حفاظت را پیدا نکرده و زنجیره تزریق در مرحله پنجم شکسته می‌شود.

۴. نتایج و بحث

در این بخش روش پیشنهادی از حیث میزان دقت در تشخیص رفتار بدخواهانه تزریق کد و کتابخانه و میزان موفقیت در ایجاد مصونیت در مقابل اینگونه حملات حملات توسط تزریق کننده‌های موجود مورد ارزیابی و آزمون قرار گرفت است. همچنین دقت روش پیشنهادی با برترین نمونه برنامه‌های موجود مقایسه شده است.

۴-۱. سنجش و مقایسه دقت

برای انجام این آزمون ۴۷۳۴ نمونه بدافزار با توانایی تزریق کد و کتابخانه از مراجع [۳۳-۳۱] در طی ۵ سال اخیر جمع‌آوری شده است. مجموعه داده بدافزارها به‌ترتیب از سال ۲۰۱۳ تا ۲۰۱۸ از مراجع فوق الذکر قابل دریافت است. بدافزارها با ۳۱۰۵ فایل بی‌خطر ترکیب شده و جهت آزمون به ابزارهای پویش‌گر معرفی شده‌اند. بدافزارها از کلاس‌های آلوده‌کننده، جاسوس‌افزار، روت کیت و چسبنده بوده و بی‌خطرها شامل فایل‌های سیستم‌عامل ویندوز نسخه ۷ و ۸ و همچنین نرم‌افزارهای کاربردی هستند. بی‌خطر بودن نرم‌افزارهای کاربردی توسط حداقل ۳ ابزار ضدویروس تایید شده است. درصد فراوانی کلاس‌های شرکت‌کننده در داده آزمون در شکل (۹) نشان داده شده است.



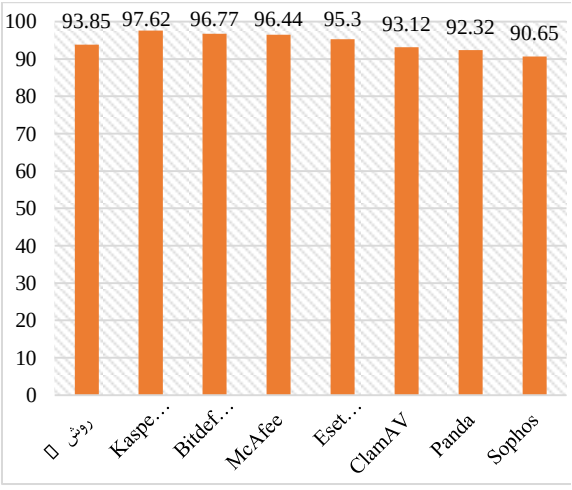
شکل ۹. فراوانی کلاس‌های موجود در مجموعه داده ارزیابی روش پیشنهادی

در ادامه ارزیابی برای مشخص شده میزان دقت روش پیشنهادی در تشخیص بدافزارها با توانایی حمله تزریق کد/کتابخانه، داده آزمون حاوی نمونه بدافزارها و فایل‌های بی‌خطر به برنامه پویش‌گر پیشنهادی که بر اساس طرح معماری ذکر شده در شکل (۸) طراحی و پیاده‌سازی شده، معرفی شده است. برای مشخص شدن آهنگ دقت روش پیشنهادی از معادله (۵) استفاده شده [۳۵] و نتایج حاصل از پویش در آن قرار داده شده‌اند. آهنگ دقت از نسبت تعداد بدافزار و فایل بی‌خطری که پویشگر پیشنهادی ماهیت آن‌ها را به‌درستی تشخیص داده، به کل نمونه‌های موجود در داده آزمون به‌دست می‌آید.

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN} \qquad (۵)$$

در این معادله TP[1] شاخص تعداد نمونه بدافزاری است که به‌درستی بدافزار تشخیص داده شده و TN[2] شاخص تعداد نمونه فایل‌های بی‌خطری است که به‌درستی بی‌خطر تشخیص داده شده است. FP[3] شاخص تعداد فایل بی‌خطری است که به اشتباه بدافزار تشخیص داده شده و FN[4] نشان‌دهنده تعداد بدافزارهایی است که به اشتباه بی‌خطر تشخیص داده شده‌اند.

بدین ترتیب آهنگ دقت برای پویش‌گر روش پیشنهادی و ۷ برنامه ضدبدافزار شناخته شده محاسبه شده است. لازم به ذکر است، مقایسه دقت روش پیشنهادی با ابزارهای ضدبدافزار دیگر در شرایط محیطی مشابه و با داده آزمون کاملاً یکسان انجام شده است. آهنگ دقت به‌دست‌آمده برای روش پیشنهادی و ۷ ابزار ضدبدافزار دیگر در شکل (۱۰) مقایسه شده است.



شکل ۱۰. مقایسه آهنگ دقت روش پیشنهادی در تشخیص رفتار تزریق کد/کتابخانه

[1] True Positive
[2] True Negative
[3] False Positive
[4] False Negative

همان‌گونه که در شکل (۱۰) مشخص است روش پیشنهادی از نظر دقت توانسته از ضدبدافزارهای مشهور مانند کِلَم اِی‌وی[1] برتر عمل کرده و با ضدبدافزارهایی نظیر ایسِت نود۳۲[2] و مَک‌آفی[3] برابری کند.

۴-۲. ضریب موفقیت سامانه خود حفاظتی

روش پیشنهادی برای تشخیص صحیح حملات تزریق کد یا کتابخانه در زمانی که سیستم آلوده است، خود باید در مواجه با این حملات مصون باشد. در این آزمون میزان موفقیت روش پیشنهادی برای ایجاد مصونیت در مقابل حملات تزریق کد و کتابخانه سنجیده می‌شود. سازوکار خود حفاظتی بر اساس روش پیشنهادی این مقاله با زبان C#.net پیاده‌سازی و بر روی پویش‌گر پیشنهادی این مقاله تعبیه شده است سپس توسط ابزارهای تزریق‌کننده کد/کتابخانه مطابق با جدول (۲) سعی در تزریق کد بدخواه به بدنه فایل باینری روی دیسک سخت و تزریق کتابخانه به حافظه فرآیند برنامه پویش‌گر (تحت حفاظت) شده است. برنامه‌های تزریق‌کننده از نمونه‌های موجود و در دسترس انتخاب شده‌اند. برخی از ابزارها امکان تزریق در بدنه فایل باینری ذخیره‌شده روی دیسک سخت (تزریق ایستا) و برخی دیگر امکان تزریق در حافظه فرآیند در حال اجرا (تزریق پویا) را دارند. در دو مورد ابزار مورد استفاده هر دو امکان تزریق به فایل اجرایی روی دیسک سخت و تزریق به حافظه اختصاص یافته به فرآیند در حال اجرا را دارد. نتایج در جدول (۲) آورده شده است.

جدول ۲ . ارزیابی ضریب موفقیت روش پیشنهادی برای خود حفاظتی در مقابل ابزارهای تزریق کد/کتابخانه

نام ابزار	ابزار [۲۹]		API Hijack		API Mon	ابزار [۳۶]
هدف حمله	فایل	فرآیند	فرآیند		فرآیند	فایل
نتیجه حمله	ناموفق	موفق	ناموفق		ناموفق	ناموفق
نام ابزار	Marshal SDK		EasyHook		Hook Tool SDK	NtCore [۳۷]
هدف حمله	فایل		فایل	فرآیند	فایل	فایل
نتیجه حمله	ناموفق		ناموفق		ناموفق	ناموفق

همان‌گونه که در جدول (۲) مشخص است روش پیشنهادی توانسته است از فرآیند و فایل‌های خود در مقابل ۸ حمله تزریق کد/کتابخانه از ۹ مورد حمله به خوبی محافظت کند که با نتیجه "ناموفق" در جدول (۲) نشان داده شده است. از تناسب میان حملات ناموفق به کل حملات صورت گرفته علیه پویش‌گر پیشنهادی، ضریب موفقیت ۸۸/۸۸ درصدی برای خود حفاظتی پویش‌گر به‌دست می‌آید.

[1] ClamAV
[2] Eset Nod32
[3] McAfee

۵. نتیجه‌گیری

بدافزارها به‌شدت در حال گسترش و پیچیده‌ترشدن هستند. از جمله روش‌های مورد استفاده بدافزارها برای مبهم‌سازی و مخفی‌سازی رفتار بدخواهانه خود، تزریق کد و کتابخانه در حافظه اجرایی دیگر فرآیندهای در حال اجرا و یا بدنه یک فایل باینری روی دیسک سخت است. این مقاله راهکاری برای تشخیص زود هنگام حملات تزریق کد/کتابخانه را با شنود روتین‌های سیستمی سطح هسته مبتنی بر قلاب IRP فراهم آورد. در ادامه این مقاله امکان خود حفاظتی در مقابل حملات مبتنی بر تزریق را تشریح کرده و طرح معماری یک سامانه پویش‌گر بر اساس روش پیشنهادی خود را ارائه کرد. در پایان مقاله دقت روش پیشنهادی خود را در تشخیص بدافزارهای کلاس آلوده‌کننده و چسبنده ارزیابی و با ابزارهای ضدبدافزار موجود مقایسه کرد همچنین ضریب موفقیت در جلوگیری از حملات تزریق کد/کتابخانه نیز سنجش شد. نتایج ارزیابی نشان‌دهنده دقت نزدیک به ۹۴ درصدی روش پیشنهادی در تشخیص حملات مبتنی تزریق کد/کتابخانه و ضریب موفقیت بالای ۸۸ درصد در خنثی‌سازی آن‌ها بود. اهمیت این پژوهش با توجه به هجوم گسترده و روز افزون بدافزارهای مبهم شده و خاص منظوره کاملاً نمایان است. همانگونه که در طول این مقاله ذکر شد روش پیشنهادی برای تشخیص زودهنگام و خنثی‌سازی حملات تزریق کد و کتابخانه بر پایه سیستم‌عامل ویندوز طراحی و پیاده‌سازی شده است. با عنایت به گسترش روز افزون بدافزارها در سیستم‌عامل‌های دیگر از جمله سیستم‌عامل آندروید پیشنهاد می‌شود راهکاری برای شناسایی و مقابله با اینگونه حملات در سایر سیستم‌عامل‌ها مورد توجه پژوهشگران قرار گیرد.

۶. مرجع‌ها

[1] Hootsuite & WAS Institute; https://wearesocial.com/blog/2018/01/global-digital-report-2018, 2018.

[2] Huang, T.; Zhao, Y. "Revolution of Securities Law in the Internet Age: A Review on Equity Crowd-Funding"; J. Comput. Law Secur. Rev. 2017, 33, 802-810.

[3] Garry, L. W. "Education and Prevention Relationships on Security Incidents for Home Computers"; J. Comput. Inform. Syst. 2015, 55, 29-37.

[4] Han, L.; Liu, S.; Han, S.; Jia, W.; Lei, J. "Owner Based Malware Discrimination"; J. Future Gener. Comput. Syst. 2018, 80, 496-504.

[5] Vidal, J. M.; Sandoval Orozco, A. L.; García Villalba, L. J. "Alert Correlation Framework for Malware Detection By Anomaly-Based Packet Payload Analysis"; J. Netw. Comput. Appl. 2017, 97, 11-22.

[6] McAfee Report "Infographic: The State of Malware"; http://www.mcafee.com/in/security-awareness/articles/state-of-malware-2013.aspx, Accessed 2014.

[7] AV-Test Security Institute "Malware Statics and Trends Report"; https://www.av-test.org/en/statistics/malware, 2016-2018.

[8] Nayeem, Kh.; Johari, A.; Adnan, Sh. "Defending Malicious Script Attacks Using Machine Learning Classifiers"; Wirel. Commun. Mob. Com. 2017.

[9] Kaspersky Report; https://usa.kaspersky.com/about/press-releases/2016_kaspersky-lab-number-of-the-year-2016-323000
-pieces-of-malware-detected-daily, 2017.

[10] Yan, J.; Qi, Y.; Rao, Q. "Detecting Malware with an Ensemble Method Based on Deep Neural Network"; Secur. Commun. Netw. 2018 (doi:10.1155/2018/7247095).

[11] Seo, S. H.; Gupta, A.; Mohamed Sallam, A.; Bertino, E.; Yim, K. "Detecting Mobile Malware Threats to Homeland Security through Static Analysis"; J. Netw. Comput. Appl. 2014, 38, 43-53.

[12] Arshad, S.; Shah, M. A.; Wahid, A.; Mehmood, A.; Song, H.; Yu, H. "SAMADroid: A Novel 3-Level Hybrid Malware Detection Model for Android Operating System"; IEEE Access 2018, 6, 4321-4339.

[13] Du, Y.; Wang, J.; Li, Q. "An Android Malware Detection Approach Using Community Structures of Weighted Function Call Graphs"; IEEE Access 2017, 5, 17478-17486.

[14] Rudd, E. M.; Rozsa, A.; Günther, M.; Boult, T. E. "A Survey of Stealth Malware Attacks, Mitigation Measures, and Steps toward Autonomous Open World Solutions"; IEEE Commun. Surv. Tutor. 2017, 19, 1145-1172.

[15] Gandotra, E.; Bansal, D.; Sofat, S. "Malware Analysis and Classification: A Survey"; J. Inf. Secur. 2014, 5, 56-64.

[16] Javaheri, D. "A Solution for Recognition and Confronting of Obfuscation and Stealth Techniques of Behavior in Spywares"; Ph.D. Thesis, Islamic Azad University, Science and Research Branch, Tehran, Iran, 2018 (In Persian).

[17] Javaheri, D. "Detection and Behavioral Analysis of Modern Malwares"; Olom Rayaneh Publications, Iran, 2017 (In Persian).

[18] OWASP Security Institute; https://www.owasp.org/ index.php/Category:OWASP_Top_Ten_Project, 2018.

[19] Javaheri D.; Parsa S. "Protection of Operation System against Spywares and Their Diversion"; J. Adv. Defence Sci. & Technol. 2014, 5, 171-181.

[20] Alam, Sh.; Horspool, R. N.; Traore, I.; Sogukpinar, I. "A Framework for Metamorphic Malware Analysis and Real-Time Detection"; Comput. Secur. 2015, 48, 212-233.

[21] Wang, P.; Wang, Y. "Malware Behavioral Detection and Vaccine Development by Using a Support Vector Model Classifier"; J. Comput. Syst. Sci. 2015, 81, 1012-1026.

[22] Javaheri, D.; Parsa, S. "A Malware Detection Method Based on Static Analysis of a Portable Executable Structure"; J. Adv. Defence Sci. & Technol. 2014, 5, 187-201.

[23] Liu, L.; Wang, B. Sh.; Yu, B.; Zhong, Q. X. "Automatic Malware Classification and New Malware Detection Using Machine Learning"; Front. Inf. Technol. Electron. Eng. 2017, 18, 1336–1347.

[24] Mohaisen, A.; Alrawi, O.; Mohaisen, M. "AMAL: High-Fidelity, Behavior-Based Automated Malware Analysis and Classification"; Comput. Secur. 2015, 52, 251–266.

[25] Hansen, S.; Larson, M. L.; Stevanovic, M.; Pedersen, J. M. "An Approach for Detection and Family Classification of Malware Based on Behavioral Analysis"; Int. Conf. on Computing, Networking and Communications, 2016.

[26] Imran, M.; Afzal, M. T.; Qadir, M. A.; Xiao, Zh.; Li, K. "Malware Classification using Dynamic Features and Hidden Markov Model"; J. Intell. Fuzzy Syst. 2016, 31, 837.

[27] Das, S.; Liu, Y.; Zhangy, W.; Chandramohan, M. "Semantics-based Online Malware Detection: Towards Efficient Real-time Protection against Malware"; IEEE Trans. Inf. Forensic Secur. 2016, 11, 289-302.

[28] Javaheri, D.; Hosseinzadeh, M. "A Framework for Recognition and Confronting of Obfuscated Malwares Based on Memory Dumping and Filter Drivers"; Wirel. Pers. Commun. 2018, 98, 119-137.

[29] Gouran Orimi, A. "Provide an Optimal and Transparent Framework for Automatic Analysis of Malware"; M.Sc. Thesis, Iran University of Science and Technology, Tehran, 2014 (In Persian).

[30] Mohammadzadeh Lajevardi, A. "Design and Implementation of a Behavior-Based Method for Malware Detection"; M.Sc. Thesis, Iran University of Science and Technology, Tehran, 2013 (In Persian).

[31] Adminus Malware Database; http://www.adminus.net, 2017-2018.

[32] Virus Share Malware Database; http://www.virusshare .com, 2016- 2017.

[33] Virus Sign Malware Data Base; http://www.virussign .com, 2013-2016.

[34] Zaki, M. J.; Wagner M. J. "Data Mining and Analysis: Fundamental Concepts and Algorithms"; Cambridge University Press, 2014, 243-339.

[35] Conway, D.; Myles, W. J. "Machine Learning for Hackers"; O`Reilly, 2012.

[36] Salmani Balu, A. "Design and Implementing a Solution for Detection and Disinfection of Injected Code"; M.Sc. Thesis, Islamic Azad University, Shabestar Branch, East Azerbaijan, Iran, 2014 (In Persian).

[37] NTCORE Injector Stub; http://www.ntcore.com/files/ inject2exe.htm, 2018.