فصلنامه علمی-ترویجی پدافند غیرعامل

سال دهم، شماره ۱، بهار ۱۳۹۸، (پیاپی ۳۷): صص ۵۸-۴۵

بهبود روش OmniUnpack جهت بازگشایی عمومی فایل اجرایی قابل‌حمل با ردیابی صفحات حافظه

یوسف شکوری[۱]، سعید پارسا[۲*]

تاریخ دریافت: ۱۳۹۶/۱۰/۳۰

تاریخ پذیرش: ۱۳۹۷/۰۷/۲۵

چکیده

تحلیل‌گران در گذشته جهت تشخیص بدافزار و تحلیل رفتار فایل اجرایی از مقایسه امضای فایل استفاده می‌کردند. نویسندگان بدافزارهای پیشرفته و جدید برای دور زدن بررسی امضا از روش‌های مبهم‌سازی جهت پنهان‌سازی اطلاعات استفاده کردند که بیشترین، مهم‌ترین و کارآمدترین روش مبهم‌سازی، بسته‌بندی کردن است. این روش بدون اینکه به رفتار فایل اجرایی اصلی صدمه‌ای بزند، *ترتیب* کدهای آن را به‌هم ریخته، رمزگذاری کرده و حتی کد را فشرده می‌کند و کد اصلی تا زمانی که اجرا نشده مبهم می‌ماند. روش‌هایی که هم‌اکنون برای بازگشایی این‌گونه فایل‌ها استفاده می‌کنند اغلب روش‌هایی هستند که به‌صورت خاص به‌ازای هر نوع بسته‌بندی‌کننده بازگشایی‌کننده مخصوص آن فایل را ایجاد می‌کنند. روش‌های دیگری نیز همچون Renovo، OmniUnpack برای بازگشایی وجود دارند که به‌عنوان بازگشایی‌کننده‌های عمومی شناخته می‌شوند و در واقع ضعف روش‌های قبلی در رابطه با نیاز به دانش از نوع بسته‌بندی‌کننده را پوشش می‌دهند، اما مشکل اصلی آن‌ها یافتن نقطه ورود اصلی برنامه یا همان انتهای بخش بازگشایی است. در اینجا برای برطرف کردن این مشکل روشی ارائه شد که با استفاده از ردیابی صفحات حافظه و پیگیری صفحات نوشته‌شده و سپس اجراشده این نقطه را شناسایی می‌کند و سپس از آن ناحیه فایل جدیدی که بازگشایی شده است ساخته می‌شود تا اولاً نیازی به دانش از نوع بسته‌بندی‌کننده وجود نداشته باشد و دوما برای بسته‌بندی‌کننده‌هایی که در آینده ایجاد می‌شوند نیز بتواند مورد استفاده قرار گیرد. در نهایت در بخش ارزیابی نشان داده خواهد شد که درصد بسیار بالایی از بسته‌بندی‌کننده‌های فعلی را می‌توان با آن بازگشایی نمود (بالای ٪۹۰) و در موتور ضد بدافزارها از آن استفاده نمود.

کلیدواژه‌ها: بازگشایی فایل اجرایی، بسته‌بندی فایل اجرایی، ردیابی صفحات حافظه، تحلیل ایستا، تحلیل پویا، رفتار فایل اجرایی

۱- دانشجوی کارشناسی ارشد، دانشگاه آزاد شبستر

۲- دانشیار، دانشگاه علم و صنعت ایران- (Parsa@iust.ac.ir) - نویسنده مسئول

۱- مقدمه

هدف اصلی بازگشایی فایل اجرایی، درنهایت دست‌یابی به فایل اولیه قبل از بسته‌بندی[1] شدن است، یعنی دقیقاً موقعی که فایل جهت اجرا بر روی سیستم‌عامل پیکربندی‌شده، بدون این‌که هیچ‌گونه روش بسته‌بندی یا مبهم‌سازی[2] یا هرگونه تغییرات دیگری جهت فشرده‌سازی[3] آن انجام شود. بعد از این مرحله است که عملیات زیادی را می‌توان بر روی فایل انجام داد، آن را تغییر داد و به کد منبعش دسترسی پیدا کرد، رفتار آن را به شکل صحیح و قابل اطمینان بیشتری مورد تجزیه‌وتحلیل قرار داده و آن را در اختیار موتور جستجوگر نرم‌افزارهای جستجو کننده بدافزار قرار داد. از اهداف دیگر بازگشایی فایل اجرایی بخصوص بازگشایی[4] عمومی، یافتن روشی جامع و کامل برای بازگشایی انواع مختلف بسته‌بندی کننده‌های موجود و حتی بسته‌بندی‌کننده‌های جدیدی که روزبه‌روز در حال پدیدارشدن هستند، است تا بتوان به‌عنوان مثال در برنامه‌های ضد بدافزار و بدافزار از آن برای شناسایی فایل‌های مخرب استفاده کرد و بتوان به امضاء فایل‌های بدافزار دست پیدا کرد و همچنین دیگر نیازی نباشد تا برای هر روش بسته‌بندی‌کننده، روش خاص و مجزا و روشی دستی با استفاده از انواع ابزارها، اعمال شود.

مسئله اصلی این است که چگونه می‌توان یک روش جامع و کلی برای بازگشایی فایل‌هایی که بسته‌بندی شده‌اند و کد اصلی آن‌ها با روش‌هایی، نامفهوم و مبهم شده‌اند را پیدا کرد، طوری که هیچ اطلاعی از نوع روش بسته‌بندی‌کننده، نداشته باشیم. درنهایت با رسیدن به این روش می‌توان رفتار آن فایل را تحلیل و بررسی کرد تا در صورت احتمال ایجاد خرابکاری توسط آن و ایجاد صدمه و آسیب‌های احتمالی و از بروز خساراتی که بعضاً جبران‌ناپذیر می‌شوند، جلوگیری کرد.

تحلیل‌گران بدافزارها و افرادی که در حال تلاش برای بازگشایی بسته‌بندی هستند، روش‌هایی را ابداع کرده‌اند، از جمله این‌که ابتدا نوع روش بسته‌بندی را تشخیص داده و سپس به‌ازای هرکدام از آن‌ها روش مخصوص بازگشایی آن را استفاده می‌کند، یا روشی که از طرز به‌هم‌ریختگی و یا فراوانی توزیع بایت‌ها جهت بازگشایی استفاده کرده‌اند و بعضی از آن‌ها نیز به دنبال ردیابی اجرا بودند تا لحظه رسیدن به نقطه انتهای بازگشایی را به‌دست بیاورند. همچنین تعدادی نیز از روش‌های ترکیبی از موارد ذکرشده استفاده کرده‌اند [۱-۴].

روش‌های موجود برای تحلیل خودکار دارای نقاط ضعف متعددی هستند. چالش اصلی مطرح‌شده در اینجا ابتدا یافتن نقطه ورودی اصلی یا OEP و انتهای بخش بازگشایی است و در ادامه آن چگونه تصویر لحظه‌ای از فایلی که الان به‌صورت کد باینری درون حافظه بارگذاری شده و در انتهای بازگشایی هست، تهیه شود. مسئله مهم دیگری که در اینجا مورد تأکید است، این است که چگونه می‌توان جدول آدرس ورودی (IAT) را دوباره ایجاد کرد تا فایل موقتی که ذخیره شده بتواند با استفاده از این جدول بر روی حافظه توسط سیستم‌عامل بارگذاری شود. چالش بعدی و مشکلی که در اینجا مطرح می‌شود، چگونگی بازگشایی فایل بدون نیاز به دانش نوع بسته‌بندی‌شده است، یعنی این‌که بدون نیاز به دانش درباره روش مورد استفاده بسته‌بندی‌کننده، بسته‌بندی فایل بازگشایی به هر ترتیبی صورت گیرد؛ که در صورت توانایی اعمال این روش دیگر نیازی نیست به بسته‌بندی‌کننده‌های جدید تا تحقیقات مخصوص آن بسته‌بندی برای بازگشایی صورت گیرد.

روش پیشنهادی ما در اینجا یک روش ردیابی پیوسته را ارائه می‌کند که اجرا در تمام آن دست نخورده است (اجرا نمی‌شود) و با یک خروج اضطراری روند بازگشایی را بهبود می‌دهد. اگر تمام دسترسی‌های صفحه-حافظه مدنظر قرار داده نشود، کارایی می‌تواند افزایش یابد. کافی است اولین دستیابی حافظه در یک گام بی‌نهایت از دسترسی‌های هم نوع مشاهده شود. برای مثال تنها اولین نوشته صفحه مفید است (به درد ما می‌خورد)، نوشته‌های بعدی در صفحات مشابه در نتایج الگوریتم خیلی مؤثر نیست و می‌تواند نادیده گرفته شود؛ بنابراین این روش بازگشایی قصد دارد خیلی جامع باشد طوری که کدهای بسته‌بندی‌شده را با هر الگوریتم دلخواهی که به هر تعداد بار بکار گرفته‌شده باشد، حمایت و پشتیبانی می‌کند.

این مقاله در شش بخش سازمان‌دهی شده است. در بخش اول، ابتدا مقدمه و سپس در بخش دوم، کارهای انجام‌شده قبلی در زمینه بازگشایی فایل‌های اجرایی، انواع بسته‌بندی‌کننده‌ها و انواع روش‌های بازگشایی مطرح، مورد بررسی قرار گرفته است. در بخش سوم، راه‌کار پیشنهادی همراه با طرز کار و الگوریتم آن ارائه و شرح داده شده است. در بخش چهارم، ارزیابی کارایی و نتایج به‌دست‌آمده از راه‌کار پیشنهادی ارائه گردید؛ و در بخش پنجم، نتیجه‌گیری مقاله و کارهای آتی ارائه شده است.

۲- کارهای مرتبط

بسته‌بندی در نرم‌افزارهای قانونی به‌منظور کاهش حجم فایل و همچنین محافظت از لو رفتن ایده کد نوشته‌شده استفاده‌شده است. نویسندگان بدافزار از بسته‌بندی برای جلوگیری از ردیابی و تشخیص بدافزار بر پایه امضای برنامه استفاده می‌کنند. روش‌های مطرح‌شده

1-Packing
2-Ambiguity
3-Compressing
4-Unpacking

در این زمینه در هفت بخش، دسته‌بندی شده‌اند که در ادامه توضیح داده می‌شود.

۲-۱- بازگشایی دستی

بازگشایی دستی ازجمله روش‌هایی است که برای دست‌یابی به بخش بازگشایی فایل اجرایی مورد استفاده قرار می‌گیرد. در این روش به‌طور معمول بازگشایی‌کنندگان از ابزارهایی همچون SoftIce و Ollydbg برای تجزیه و تحلیل لایه‌های رمزگذاری[1]شده توسط بسته‌بندی‌کننده و الگوریتم‌های از فشرده خارج‌سازی و بازیابی دستی فایل اصلی، استفاده می‌کنند. واقع در این روش ما فایل بسته‌بندی‌شده را با استفاده از ابزارهایی مثل ollydbg بارگذاری کرده و با اجرای مرحله‌ای فایل موردنظر به دنبال نقطه ورود اصلی برنامه یعنی همان نقطه‌ای که کدهای اجرای برنامه اصلی وجود دارند، می‌گردیم. از مزایای این روش می‌توان به کشف غالب خطاهای مخفی و کشف‌نشده برنامه‌ها اشاره کرد و از معایب آن نیز این است که این فرایند، زمان صرف کرده و نیاز به درک عمیق از برنامه‌نویسی هسته (هسته سیستم‌عامل) و اسمبلی دارد [۶-۵].

۲-۲- روش نرمال‌سازی

ابتدا اجرای برنامه را در یک محیط کنترل‌شده جهت تشخیص جریان کنترل دستوراتی که کنترل را درون ناحیه کد تولیدشده می‌برند، برده می‌شود. به‌طور مثال در اجرای برنامه یک نمونه‌ساز، تمام نوشته‌های حافظه جمع‌آوری می‌شود (حفظ فقط آدرس‌هایی که تازه مقداردهی شده‌اند) و جریان اجرا ردیابی می‌شود. اگر برنامه تلاش کند تا کد را از حافظه‌ای که از قبل نوشته‌شده اجرا کند، آدرس مقصد، انتقال جریان کنترل را به‌دست خواهد آورد (مثل دستورات راه‌اندازی) و اجرا را متوقف خواهد کرد. لحظه‌ای که اجرا به مکان حافظه نوشته‌شده قبلی دست می‌یابد، به‌وسیله نمونه‌سازی برنامه و ردیابی هر دستوری که اجرا شده، می‌تواند تشخیص داده شود. دوم، با اطلاعات به‌دست‌آمده در مرحله قبلی، یک برنامه عادی‌شده (نرمال‌شده) ساخته که شامل کد تولیدشده است [۱].

مزایای این روش سربار کم و سرعت بالا است. این روش دارای عیوب اساسی است. فایل اجرایی بازگشایی‌شده آماده اجرا نیست. هرچند فایل بسته‌بندی‌شده می‌تواند بازگشایی شده و در شکل فایل اجرایی نرمال‌شده، تولید شود، اما چون فایل واقعی نیست، ازآنجایی‌که بیشتر، مبهم‌سازی بسته‌بندی، آن را با بارگذارنده پویای دستی جایگزین می‌کند، جدول ورودی‌ای که فراخوانی DLLها و APIها لیست می‌کند، ممکن است توسط برنامه پیدا نشود.

۲-۳- روش رنوو

این روش سایه‌ای از حافظه فضای حافظه تحلیل‌شده را نگه می‌دارد، اجرای برنامه را می‌بیند و اگر دستورات تولیدشده به‌تازگی اجرا شده‌اند، آن را تشخیص می‌دهد. سپس کد و داده ایجادشده را خارج می‌کند. فرض کنید هیچ چیزی درباره فشرده‌سازی باینری و روش‌های رمزگذاری وجود نداشته باشد، آن ابزاری را برای استخراج کد و اطلاعات مخفی فراهم می‌کند که در برابر روش‌های ضد مهندسی معکوس قوی است. بعد از اینکه فایل اجرایی بسته‌بندی‌شده، اجرا شد، زیربرنامه رمزگشایی ضمیمه‌شده آن، رویه‌های دگردیسی را بر روی داده بسته‌بندی‌شده فراهم می‌کند (لایه‌های مخفی نیز گفته می‌شود) و کد و داده اصلی بازیابی می‌شود. بعد از این، زیربرنامه رمزگشایی زمینه اجرا را برای کد برنامه اصلی فراهم می‌کند تا اجرا شود. این کارها، ثبات‌های پردازنده و شمارنده برنامه را برای نقطه ورود کد جدید تولیدشده در حافظه آماده می‌کند [۷]. از مزایای این روش این است که اول هیچ فرضی درباره روش بسته‌بندی به‌جز اینکه کد مخفی اصلی بالاخره در زمان اجرا نوشته‌شده و اجرا خواهد شد، ندارد؛ بنابراین این روش قادر به دست گرفتن هر ترتیبی از روش‌های بسته‌بندی به‌کار گرفته‌شده برای باینری‌ها است. دوما این روش می‌تواند ناحیه دقیق حافظه کد یا داده ایجادشده را در زمان اجرا تشخیص دهد. ازآنجایی‌که اطلاعات درباره نوشته‌های حافظه در سطح بایت نگه‌داری می‌شوند، امکان دارد داده و کد تولیدشده جدید را استخراج کند. در آخر، این روش به هیچ‌کدام از اطلاعات بر روی بخش داده و کد باینری تکیه نمی‌کند. ولی از عیوب مهم آن این است که از نقاط ضعف نمونه‌سازی رنج می‌کشد و می‌تواند به‌راحتی توسط روش‌های ضد نمونه‌سازی دور زده شود. بسته‌بندی‌کننده‌ها همچنین می‌توانند از روش ضد دامپینگ حافظه که بخش کد را بعد از این‌که اجرا شد فوراً حذف می‌کند، بهره ببرند و در نتیجه کار بازگشایی را با مشکل روبرو کنند.

۲-۴- روش بی‌نظمی

با تکیه بر این اصل که بی‌نظمی فضای حافظه به‌طور پیوسته در حال تغییر است تا زمانی که دستورات بسته‌بندی‌شده درون حافظه بازگشایی شوند، فایل‌های دودویی با بی‌نظمی و پراکندگی بایت‌های زیاد اغلب نشان‌دهنده فایل فشرده‌شده یا رمزگذاری شده است. ابتدا با کمک تحلیل بی‌نظمی می‌توان لحظه تمام شدن بازگشایی را مشخص کرد. روش ارائه‌شده، عامل مؤثری در پیدا کردن نقطه ورود اصلی است. رابطه شانون برای اندازه‌گیری میزان پراکندگی اطلاعات به‌صورت زیر تعریف شده است:

$$H(x) = \sum_{i=1}^{n} p(i).\log_b p(i)'$$

[1] Encoded

در رابطه فوق H(X) مقدار پراکندگی اندازه‌گیری شده، p(i) احتمال رخداد x در i امین مرحله است. پایه لگاریتم می‌تواند هر عدد حقیقی بزرگ‌تر از یک باشد. هرچند در حالت کلی ۲ و ۱۰ و عدد اویلر e انتخاب است. مفهوم اصلی نتیجه‌شده از تفاوت مقادیر پراکندگی اندازه‌گیری‌شده بین دستورات بسته‌بندی‌شده و بازگشایی‌شده، بیانگر رفتار فرایند بازگشایی است. در اصل ما فایل اجرایی بسته‌بندی‌شده را اجرا می‌کنیم و اجازه می‌دهیم تا فرایند بازگشایی انجام شود. در طول این فرایند دستورات بسته‌بندی‌شده به‌وسیله یک بخش کوچک از وضعیت فشرده خارج و بازگشایی می‌شوند. سرانجام، انتهای بخش بازگشایی‌شده را می‌توان از طریق ردیابی پایان تغییرات بی‌نظمی تشخیص داد. از مزایای این روش امکان کشف دستورات پنهان تولیدشده در طول فرایند بازگشایی است. از جمله معایب آن نیز این است که بخش بازگشایی به‌طور مکرر دستورات پنهان را مشخص می‌کند که این عمل باعث افزایش زمان تحلیل می‌شود.

۲-۵- روش ترکیبی

اگر فایل اجرایی با بسته‌بندی‌کننده پیچیده بسته‌بندی شود، استفاده ترکیبی از نمونه‌سازی و زیر برنامه‌های بازگشایی برای شناخت بسته‌بندی کننده بسیار مناسب خواهد بود به‌طور مثال نمونه‌سازی بر اساس بازگشایی که در این صورت بازگشایی پویا خیلی کند بوده و یک‌راه حل ترکیبی راه‌حلی است که از مزیت‌های هردوی آن‌ها استفاده می‌کند. بسته‌بندی‌کننده‌های ایستا و ثابت، کد چند دگردیسی را به‌کار نمی‌برند؛ و به‌جای آن، نمونه‌ساز عمومی کند را به‌کار می‌گیرد. مزیت این روش اگرچه استفاده از زیربرنامه‌های مشخص است که روشی مناسب است اما عیب آن این است که ترکیب آن با نمونه‌سازی، پیچیدگی اضافی بر روی سیستم خواهد داشت. مزیت دیگر این روش، جامع بودن آن به‌وسیله نمونه‌ساز عمومی و سرعت همچون بازگشایی مشخص مستقل از کد نمونه‌سازی‌شده، است. آن به‌راحتی قابل بسط است و می‌تواند پله‌پله پیاده‌سازی شده و گسترش یابد، (همچون آماده‌سازی تعداد کمی از زیربرنامه بازگشایی مشخص نوشته‌شده) و با زمان، نتایج می‌تواند در حوزه وسیع‌تر به‌کار گرفته شود [۸].

۲-۶- روش چندگانه

اساس رفتار این روش استفاده از ترکیبی از تحلیل ایستا و پویا است تا فرایند استخراج کد مخفی بدافزار بسته‌بندی‌شده را خودکار کند. اهمیت این روش در نتایج اجرای بازگشایی (مثل اجرای کد تولیدشده زمان اجرا) نسبت به ساز و کار بازگشایی استفاده‌شده، است. این روش روش‌های دیگر را جابجا می‌کند، مثل دانش قبلی درباره بسته‌بندی‌کننده یا برنامه‌نویسی صریح رفتار سامانمند به‌دست‌آمده، که تمام نمونه‌ها در یک کلاس بازگشایی موردنیاز نیست. این روش ابتدا یک دید ایستا از برنامه بسته‌بندی‌شده (مثلاً ترتیب کد برنامه هیچ کدی را در زمان اجرا فراهم نمی‌کند) در حافظه با استفاده از تحلیل ایستا ایجاد می‌کند. مدل ایستای کد بعداً به موتور تحلیل پویا ارسال می‌شود (مثل نمونه‌ساز یا هر محیط مجازی دیگر). این تحلیل پویا تحلیل زمان اجرای معمول متفاوت است به‌طوری‌که اگر ترتیب اجرا مطابق با هر یک از بخش‌های مدل ایستا باشد، قادر است تا بررسی کند. تا زمانی که بخش کوچک (زیر برنامه بازیابی) اجرا شد، کد از ترتیبی همچون ترتیب در دید ایستای بخش کوچک پیروی خواهد کرد. بعد از اجرای هر دستورالعمل، فضای اجرا با مدل ایستا مقایسه می‌شود. نقطه‌ای از زمان اجرا وقتی‌که کد از دید ایستا منحرف شد، نشان می‌دهد که کد بازگشایی شده است (از آنجایی‌که ترتیب کد بازگشایی‌شده در مدل ایستا موجود نیست). این شرط توقف است و کد بازگشایی‌شده فایل اجرایی را تحت تجزیه‌وتحلیل اجرا فراهم می‌کند. این روش از واقعیت کد مخفی‌شده مبهم‌شده استفاده می‌کند که در مدل ایستا موجود نیست. بنابراین، این روش حوزه وسیعی را برای تحلیل ایستا می‌دهد و سازوکاری که مستقل از سازوکار بسته‌بندی است را فراهم می‌کند.

این تمام آن چیزی نیست که بازگشایی‌کننده انجام می‌دهد، همیشه موردی که ترتیب کد را در مدل ایستا غیب و مخفی کند نیست. Dllهای بارگذاری‌شده در طول اجرای باینری ویندوز، در اجرای کدی که در مدل ایستا موجود نیست نتیجه می‌شود؛ بنابراین، هر وقت DLLای بارگذاری شد، مکان اسکان آن در حافظه نوشته می‌شود. در مدت اجرای یک گام، شمارنده برنامه به‌طور پیوسته با تمام نواحی شناخته‌شده حافظه مقایسه می‌شود. اگر شمارنده برنامه به‌جایی از حافظه اشاره می‌کند که توسط Dllای اشغال‌شده، آدرس برگشتی پشته را خوانده و یک نقطه انفصال در آنجا درج می‌کند، و اجرای گام‌به‌گام اجازه ادامه کار بعد از برگشت از فراخوانی‌اش را می‌دهد.

برای بررسی افزایش پیچیدگی برنامه در طول لایه‌های بسته‌بندی‌شده چندگانه، روش چندبازگشایی می‌تواند مکرراً گسترش یابد. هر زمان برآیند کد مخفی حاصل شد، مدل ایستای کد، تولیدشده است. اگر در هر مرحله گام باینری اجراشده کد با هیچ‌کدام از دیدگاه ایستا مطابقت نکرد، دوباره از سر گرفته می‌شود. کد نهایی بازگشایی‌شده مدل ایستایی است که مجموعاً شامل ترتیب اجرای زمان اجرا است. این روش بیشتر شبیه تجهیز ابزارهایی است که برای پردازش بدافزار شفاف نیست؛ بنابراین امکان اجراشدن یک نمونه از بدافزار وجود دارد که در این صورت رفتارش را برای

جلوگیری از استخراج کد بازگشایی‌شده‌اش تغییر خواهد داد (مثل متوقف شدن بجای مخفی کردن کدهای تولیدی) [۹].

۲-۷- روش بازگشایی پویا

روش دیگری که هم‌اکنون در موتورهای جستجوگر آنتی‌ویروس‌ها به‌کار گرفته می‌شود، بازگشایی پویا است، برنامه‌های آنتی‌ویروس برای خودکار کردن تشخیص بسته‌بندی‌کننده، معمولاً آن دسته از بسته‌بندی‌کننده‌های ایستا را گسترش می‌دهند که زیربرنامه‌های اختصاصی برای فشرده خارج‌سازی/رمزگشایی فایل‌های بسته‌بندی‌شده دارند و این کار به‌وسیله تعیین بسته‌بندی‌کننده‌ها بدون اجرای واقعی برنامه‌های مشکوک صورت می‌گیرد. از آن بسته‌بندی‌کننده‌های ایستا می‌توان به نرم‌افزارهای اضافی کمکی بازگشایی UPX و Upack اشاره کرد. بازگشایی پویا برای بازگشایی فایل‌های بسته‌بندی‌شده‌ای که معلوم نیست با چه روشی بسته‌بندی شده‌اند خیلی مفید است، ولی توسعه‌دهندگان ویروس‌ها می‌توانند با بسته‌بندی‌کننده‌های دستی و غیرقابل‌پیش‌بینی، آن را فریب دهند. نرم‌افزار IDA Pro یک اشکال‌زدا بر اساس بازگشایی کلی را فراهم می‌کند. صرف‌نظر از بعضی استثنائات، بیشتر برنامه‌های مبهم شده نیاز به یک پرش طولانی (مشترک در بیشتر برنامه‌های مبهم شده) دارند تا جریان اجرایی را از بخش بسته‌بندی‌شده به بخشی که نقطه ورود اصلی برنامه قرار دارد، منتقل کند. اگر موتور بازگشایی عمومی بتواند به دستورات پرش (دستورات پرش مشترک) دست یابد، موتور آنتی‌ویروس می‌تواند با استفاده از روش‌های ابداعی مشخص کند که نقطه ورود اصلی برنامه کجا قابل دست‌یابی است. باید بدانیم در فایل بسته‌بندی‌شده دونقطه ورود داریم، نقطه اول همان نقطه‌ای است که آدرس آن برای اجرا به سیستم‌عامل فرستاده می‌شود و همان ابتدای بخش بسته‌بندی‌شده است و دیگری نقطه ورود اصلی که درواقع زیر برنامه‌ای که در بخش بسته‌بندی‌شده ابتدایی قرار دارد نقطه ورود را مشخص می‌کند. درواقع با پیدا کردن نقطه ورود اصلی مرحله اصلی بازگشایی فایل انجام شده است و این نقطه همان‌جایی است که کد اصلی در آنجا قرار دارد. بسته‌بندی‌کننده، زیربرنامه فشرده شدن است که فایل اجرایی را فشرده کرده و حقه‌ای برای مقایسه امضاء فایل فشرده‌شده ایجاد می‌کند. بسته‌بندی‌کننده‌ها در ابتدا ساختار داخلی اجرایی قابل‌حمل را می‌فرستند، سپس سربرگ اجرایی قابل‌حمل، جدول خروجی و جدول ورودی در ساختار جدید و کد سگمنت‌های ضمیمه‌شده را قبل از نقطه ورود اصلی برنامه که ناحیه تحتانی نامیده می‌شود را مشخص می‌کنند. به‌طورکلی یک برنامه بسته‌بندی‌شده به‌محض اجرا ابتدا برنامه را بر روی حافظه بازگشایی می‌کند، کتابخانه‌های موردنیاز را بارگذاری می‌کند و به توابع ورودی توسط جستجوی جدول آدرس ورودی فایل اجرایی دست می‌یابد. ساختار برنامه اصلی افشا شده و در این نقطه یک تصویر فوری از حافظه می‌تواند برداشته شده و در فایلی دیگر ذخیره شود، به این عمل گرفتن تصویر فوری گفته می‌شود. این فایل برداشته‌شده از حافظه می‌تواند برای تحلیل امضا توسط یک بررسی‌کننده ویروس تحلیل شود.

مزیت این روش این است که کار بازگشایی پیچیده و بحرانی می‌تواند توسط خود زیر برنامه صورت گیرد. دشوارترین بخش این روش، تعیین دقیق زمانی است که از حافظه باید تصویر یا نسخه فوری برداشته شود. اگر تصویر لحظه‌ای حافظه به‌موقع صورت نگیرد (قبل از این‌که برنامه به‌طور کامل بازگشایی شود)، کدهای مخفی دست‌نخورده به هیچ دردی (تحلیل) نخواهند خورد؛ و اگر تصویر لحظه‌ای خیلی دیر کند، برنامه اجرای بخش بازگشایی‌شده را آغاز خواهد کرد و سیستم را در برابر حمله بدافزار آسیب‌پذیر خواهد کرد [۱۰]. بعلاوه فایل اجرایی گرفته‌شده نیاز دارد تا ساختار سربرگ فایل[1] ثابت شود، ولی خود کد در شکل اصلی‌اش قابل‌رؤیت است و برای مهندسی معکوس و تحلیل پویا قابل‌استفاده است. این روش ساده برای انواع بیشتر بسته‌بندی‌کننده‌ها و رمزگذارنده‌های فایل‌های اجرایی کار می‌کند، طوری که تابع بازگشایی معمولاً کل برنامه را در آغاز کار استخراج می‌کند و با محاسبات دیگر تداخلی ندارد (ولی همیشه این‌طور نیست). بزرگ‌ترین اشکال این روش این است که فایل اجرایی باید بارگذاری شود که ممکن است همیشه پذیرفتنی نباشد، یعنی این‌که هر فایلی که می‌خواهیم بازگشایی کنیم با استفاده از این روش آن فایل حتماً باید اجرا شود که اگر کنترل آن از دستمان خارج شد ممکن است تبعات جبران‌ناپذیری داشته باشد. همچنین آن نمی‌تواند تضمین کند که برنامه قبل از هر فراخوانی تابع بدافزار متوقف خواهد شد. برای برطرف کردن مشکل اصلی این روش و مشکلات روش‌های دیگر که برای تحلیل و بازگشایی باید اجرا شوند، جهت تحلیل رفتار و تجزیه و بازگشایی این‌گونه فایل‌های بسته‌بندی‌شده از محیط‌های مختلفی می‌توان بهره برد.

با تکیه بر این اصل که بی‌نظمی فضای حافظه به‌طور پیوسته در حال تغییر است تا زمانی که دستورات بسته‌بندی‌شده درون حافظه بازگشایی شوند، فایل‌های دودویی با بی‌نظمی و پراکندگی بایت‌های زیاد اغلب نشان‌دهنده فایل فشرده‌شده یا رمزگذاری شده است. ابتدا با کمک تحلیل بی‌نظمی لحظه‌ای که بازگشایی تمام می‌شود را می‌توان مشخص نمود. روش ارائه‌شده عامل مؤثری در پیدا کردن نقطه ورود اصلی است. با توجه به این‌که این روش بر اساس امضاها کار نمی‌کند، حتی اگر بسته‌بندی‌کننده، ناشناس باشد می‌توان نقطه ورود اصلی را پیدا کرد.

اغلب تحقیقات در زمینه بازگشایی، بر روی فایل‌های بسته‌بندی‌شده اجرایی در یک محیط خاص، همچون

[1] File header

اشکال‌زداکننده‌ها یا ماشین مجازی است که جریان اجرایی را کنترل می‌کند؛ بنابراین امکان کشف دستورات پنهان تولیدشده در طول فرایند بازگشایی وجود دارد؛ که نمونه آن را می‌توان روش بازگشایی چندگانه، بازگشایی تمام جانبه، رنوو که در این پایان‌نامه نیز به آن‌ها اشاره شده، دید. یافتن نقطه ورود از نیازهای اولیه برای بازگشایی است. در تئوری اطلاعات، میزان پراکندگی، حد غیرقطعی در یک سری از واحد اطلاعات است. اطلاعات با پیروی از مراحل منطقی فشرده شده است. ابتدا برخی الگوهای تکراری در اطلاعات پیدا شده و سپس از این الگوها برای کاهش اندازه فایل استفاده شده است. تعداد الگوهای اطلاعات کاهش داده شده توسط فشرده‌سازی و یک سری از بیت‌ها زیاد قابل پیشگویی نیستند که معادل غیر اطمینان بودن است؛ بنابراین، میزان پراکندگی اطلاعات فشرده‌شده اندازه‌گیری شده خیلی فراتر از اطلاعات اصلی است. رابطه شانون برای اندازه‌گیری میزان پراکندگی اطلاعات به‌صورت زیر تعریف شده است:

$$H(x) = \sum_{i=1}^{n} p(i).\log_b p(i), \qquad (۱)$$

رابطه (۱) اندازه‌گیری میزان پراکندگی اطلاعات را نشان می‌دهد.

در این روش، یک فایل اجرایی به‌عنوان ورودی داده‌شده و اگر فایل بسته‌بندی‌شده باشد، این روش نقطه ورود اصلی را به‌عنوان نتیجه می‌دهد. ابتدا فایل اجرایی اجراشده و در حالت اجرا نگه‌داشته می‌شود تا این‌که یکی از دستورات CALLJCCJMP و یا RET بیاید. اگر به یکی از این دستورات ذکر شده رسیدیم، اجرا متوقف شده و بی‌نظمی موجود مورد تجزیه‌وتحلیل قرار می‌گیرد. این بدین خاطر است که وقتی بازگشایی تمام شد این دستورات هستند که باید مسیر اجرایی را از انتهای بخش از فشرده‌سازی، به سمت نقطه ورود اصلی هدایت کنند.

تحلیل بی‌نظمی به‌وسیله اندازه‌گیری فضای حافظه صورت می‌گیرد. اگر دستوری پیدا شد که به یک بخش کد بازگشایی‌شده پرش می‌کند، آنجاست که تصمیم می‌گیرد آیا بازگشایی تمام‌شده یا نه. مرحله بعدی تشخیص توسط نتیجه تحلیل بی‌نظمی است، اگر فایل بسته‌بندی‌شده بازگشایی‌شده است، نقطه ورود اصلی همان‌جا است ولی اگر فرایند بازگشایی کامل نشده است، فرایند متوقف‌شده، به ادامه اجرای دستورات بعدی خواهد پرداخت. موضوع دیگر درباره تغییر جریان اجرای برنامه تعداد دستورات است. بخش بازگشایی مکرراً از این دستورات دارد که باعث افزایش زمان تحلیل می‌شود که برای حل این مشکل تعداد آدرس‌های دستورات پرش که در هر بار تحلیل ملاقات می‌شوند نگهداری می‌شود، اگر به یک آدرس که قبلاً ملاقات شده و نگه‌داشته شده، رسیدیم، دیگر از آن نقطه به بعد تحلیل بی‌نظمی نادیده گرفته می‌شود.

در طول فرایند بازگشایی دستورات در یک بخش بازگشایی می‌شوند، با توجه به رابطه گفته‌شده جهت اندازه‌گیری بی‌نظمی داده‌ها، مقدار بی‌نظمی اندازه‌گیری‌شده، به حالت داده اشاره می‌کند که یا حالت بازگشایی‌شده یا بسته‌بندی‌شده یا شروع بازگشایی، در هر بخش از حافظه را بیان می‌کند. نتایج گرفته‌شده نشان می‌دهد که وقتی فایل بسته‌بندی‌شده بازگشایی می‌شود مقادیر بی‌نظمی تغییر می‌کنند. درنتیجه می‌توان مشخص کرد اگر بازگشایی تمام‌شده تحلیل بی‌نظمی انجام شود؛ درنتیجه می‌توان با این روش به‌جای تحلیل پویا، هر زمان که نیاز شد به تحلیل فایل پرداخته شود.

۲-۷-۱- روش OmniUnpack

تمام روش‌های پوشش داده‌شده محدودیت‌هایی دارند، حتی روش چندگانه بازگشایی، از اشکال‌زدایی تک‌مرحله‌ای رنج می‌کشد؛ بنابراین، یک روش جایگزین برای روش‌های ذکرشده روش OmniUnpack است [۱۹]. هدف این روش، رفع کاستی‌های سیستم موجود است. این روش یک روش عمومی و جامع برای اجرای هر نوع بسته‌بندی‌کننده و هر کد خودتغییر مناسب است که به محیط مجازی نمونه‌سازی یا اشکال‌زدایی کردن برای بازگشایی، بستگی ندارد. این روش اجرای برنامه، پیگیری نوشته‌ها و صفحات حافظه نوشته‌شده سپس اجراشده را ردیابی و ردگیری می‌کند. صفحات نوشته‌شده سپس اجراشده، نشان‌دهنده بازگشایی است اما نشان‌دهنده پایان بازگشایی نیست، طوری که در آنجا باید مراحل بازگشایی چندگانه انجام گیرد. این روش از اکتشافاتی برای بازگشایی نزدیک به بازگشایی نهایی استفاده می‌کند. برای بهبود کارایی، ردیابی سطح صفحه انجام می‌شود. وقتی بخش ناحیه تحتانی که در برنامه تعبیه شده، کد بازگشایی را در حافظه می‌نویسد، صفحه مقصد به‌عنوان صفحه قابل‌نوشتن علامت زده می‌شود ولی اجرا نمی‌شود. در انتهای مرحله بازگشایی وقتی برنامه به بعضی صفحات اجرایی دسترسی می‌یابد، نبود مجوز اجرا باعث اجرای محافظت‌شده می‌شود. اگر برنامه یک فراخوانی سیستمی خطرناکی داشته باشد، تشخیص‌دهنده بدافزار بر روی صفحات نوشته‌شده حافظه درخواست می‌دهد. اگر نتیجه تشخیص منفی بود (مثل هیچ بدافزاری پیدا نشد)، اجرا ادامه می‌یابد. دستاورد سربار کم بدین معنی است که این روش می‌تواند برای ردیابی پیوسته محصول سیستم به‌کار گرفته شود.

وقتی نگاشت آدرس فیزیکی مجازی نیاز به به‌روزرسانی داشته باشد یا وقتی‌که محافظ حافظه شکسته شد، سخت‌افزار به‌واسطه یک اجرا سیگنالی به سیستم‌عامل می‌دهد و سیستم‌عامل اجازه مراحل تعمیر حافظه را قبل از ادامه اجرا می‌دهد. ویژگی‌های موجود برای

جدا کردن اولین لحظه‌ای که صفحه نوشته شد و اولین لحظه‌ای که صفحه بعد از نوشتن اجرا شد، استفاده می‌شود.

از عیوب روش، عدم دقت کافی در پیگیری سطح صفحه است. پیگیری سطح صفحه ضمانت ردیابی را کاهش می‌دهد با اینکه این روش سربار ردیابی دسترسی به حافظه را به‌طور محسوسی کاهش می‌دهد. این روش کم‌دقت است و اغلب در تشخیص مراحل بازگشایی اشتباه می‌کند. باید هر زمان که صفحه حافظه نوشته‌شده اجرا شد، تشخیص‌دهنده بدافزار را درخواست کرد، نباید پرهزینه باشد، چون این رویداد (نوشته شده-سپس اجراشده) مکرراً رخ می‌دهد؛ بنابراین، تشخیص این‌که تصمیم بگیریم بازگشایی تمام‌شده یا نه سخت است و فقط تقریبی است. این روش عموماً فرض می‌کند که برنامه بسته‌بندی‌شده، بدافزار یا کد بدخواه است؛ بنابراین اگر فراخوانی سیستمی خطرناکی ایجاد شد، امکان فراخوانی خودکار موتور تشخیص بدافزار را فراهم می‌کند. این نشان می‌دهد که باید الگوریتم پیچیده‌ای برای تشخیص این‌که یک فراخوانی سیستمی، فراخوانی خطرناک است یا نه ساخته شود. فراخوانی سیستمی خطرناک، فراخوانی است که اجرا می‌تواند سیستم را در حالت غیر ایمن ترک کند. برای دست‌یابی بدافزار به اهدافش بدافزار با سیستم فعل و انفعالاتی دارد. در این روش به‌عنوان راه‌حل ساده، هر فراخوانی سیستمی که وضعیت سیستم‌عامل را تغییر داد خطرناک محسوب می‌شود. چون امکان مراحل بازگشایی چندگانه برای تشخیص آن استفاده می‌شود، ردیابی کردن و بررسی برنامه تنها یک‌بار در طول اجرا کافی نیست. این روش یک روش ردیابی پیوسته را ارائه می‌کند که اجرا در تمام آن دست نخورده است (اجرا نمی‌شود). این عملیات یک خروج اضطراری از دید بازگشایی سنتی و جستجو کردن به‌طور مجزا، تک مراحل فرایند تشخیص بدافزار است. همچنین اگر تمام دسترسی‌های صفحه-حافظه مدنظر قرار داده نشود، کارایی می‌تواند افزایش یابد. کافی است اولین دست‌یابی حافظه در یک گام بی‌نهایت از دسترسی‌های هم نوع را مشاهده کنیم. برای مثال تنها اولین نوشته صفحه مفید است (به درد ما می‌خورد)، نوشته‌های بعدی در صفحات مشابه در نتایج الگوریتم خیلی مؤثر نیست و می‌تواند نادیده گرفته شود؛ بنابراین، این روش بازگشایی قصد دارد خیلی جامع باشد طوری که کدهای بسته‌بندی‌شده را با هر الگوریتم دلخواهی که به هر تعداد بار به‌کار گرفته شده باشد، حمایت و پشتیبانی می‌کند [۲].

۳- روش پیشنهادی

این روش یک روش کاملاً عمومی بوده که دستورات تازه اجراشده و نوشته‌های بر روی حافظه را ردیابی می‌کند، اجرای برنامه را می‌بیند و اگر دستورات تولیدشده به‌تازگی اجرا شده‌اند، آن تشخیص می‌دهد سپس کد و داده ایجادشده را خارج می‌کند. فایل اجرایی بسته‌بندی‌شده ممکن است چندین لایه مخفی داشته باشد و حتی ایجاد آن خیلی سخت‌تر از تحلیل آن باشد؛ اما صرف‌نظر از روش بسته‌بندی و لایه‌های مخفی، کد و داده برنامه اصلی به‌صورت نامحدود در حافظه خواهد بود. به‌علاوه، اشاره‌گر دستور باید به سمت نقطه ورود اصلی برنامه کد برنامه بازیابی‌شده پرش کند که در زمان اجرا در حافظه نوشته شده است. بهره‌گیری از این خصوصیات فایل‌های اجرایی بسته‌بندی شده، یک الگوریتم عمومی برای استخراج کد اصلی مخفی‌شده و استخراج نقطه ورود اصلی برنامه از فایل بسته‌بندی شده را می‌تواند ایجاد کند، چه دستور جاری در زمان اجرا ایجاد شده باشد و چه بعد از این‌که کد برنامه بارگذاری شد.

از مزیت‌های این روش می‌توان به سه مورد اشاره نمود: اول اینکه هیچ فرضی درباره روش بسته‌بندی به‌جز این‌که کد مخفی اصلی بالاخره در زمان اجرا نوشته شده و اجرا خواهد شد، ندارد، بنابراین، این روش قادر به‌دست گرفتن هر ترتیبی از روش‌های بسته‌بندی به‌کار گرفته‌شده برای باینری‌ها است. دوما این روش می‌تواند ناحیه دقیق حافظه کد یا داده ایجاد شده را در زمان اجرا تشخیص دهد، ازآنجایی‌که اطلاعات درباره نوشته‌های حافظه در سطح بایت نگهداری می‌شوند، امکان دارد داده و کد تولیدشده جدید را استخراج کند. در آخر، این روش به هیچ‌کدام از اطلاعات بر روی بخش داده و کد باینری تکیه نمی‌کند، وقتی یک کد اجرایی تحلیل میگردد، آن در یک محیط نمونه‌سازی شده اجرا می‌شود. محیط نمونه‌سازی‌شده شبیه‌سازی دستورات پردازنده را در دستورات مخصوصی که نوشته‌های حافظه را فراهم می‌کند آسان می‌کند.

وقتی بازگشایی‌کننده فایل بسته‌بندی‌شده، بازگشایی برنامه تعبیه‌شده را کامل کرد، جدول آدرس ورودی را تنظیم می‌کند، پشته را رها کرده و کنترل اجرا را به نقطه ورود برنامه تعبیه‌شده منتقل می‌کند؛ بنابراین، شرایط لازم برای اجرای فایل بسته‌بندی‌شده جهت دست‌یابی به انتهای بازگشایی موارد زیر هستند:

- کنترل به صفحه ایجادشده یا تغییریافته پویا منتقل شود.
- پشته شبیه وقتی باشد که برنامه درون حافظه بارگذاری شد.
- آرگومان‌های ورودی خط فرمان به‌طور صحیح بر روی پشته تنظیم شوند.

در ابتدای بارگذاری، فایل تمام صفحات را به‌عنوان صفحات اجرایی ولی غیرقابل نوشتن علامت‌زده و اجرای فایل آغاز می‌شود. اگر در طول اجرا خطای نوشتن بر روی یکی از صفحاتی که به‌عنوان غیرقابل‌نوشتن علامت زده شده، روی دهد، آن‌گاه این صفحه به‌عنوان صفحه کثیف علامت زده و آن را به‌عنوان غیرقابل اجرا و قابل نوشتن علامت زده و ادامه داده می‌شود. اگر در ادامه خطای اجرا رخ دهد، یعنی صفحه‌ای جهت اجرا مورد دست‌یابی قرار گیرد

که به‌عنوان غیرقابل اجرا علامت زده شده است، این روش بخش جستجوی بررسی تمام تصویر حافظه فراخوانی می‌کند و اگر بررسی نتیجه رسیدن به انتهای بخش بازگشایی، منفی باشد، همین روند درون صفحات اجرا پذیر غیرقابل نوشتن ادامه می‌یابد. توجه داشته باشید که تصویر کل حافظه به‌عنوان یک فایل بررسی‌شده توسط جستجوگر بدافزار ارائه می‌شود. البته این متفاوت از روش بازگشایی OmniUnpack است که تنها صفحات کثیف را مورد بررسی قرار می‌دهد. برای یک فایل دودویی بسته‌بندی نشده، به خاطر این‌که هیچ‌گونه کد صفحه‌ای در طول اجرا کثیف یا مورد تغییر قرار نگرفته است، پس هیچ خطایی رخ نداده و هیچ بررسی‌کننده‌ای در زمان اجرا صدا زده نخواهد شد؛ بنابراین، سربار این روش برای فایل‌های بسته‌بندی نشده بسیار ناچیز است؛ به‌عبارت‌دیگر این روش بهترین کاریی را برای فایل‌های دودویی بسته‌بندی نشده دارد، زیرا بسته به تعداد دفعات کنترلی که در برنامه برای صفحات جدید ایجادشده صورت می‌گیرد، به همان نسبت سربار بالا خواهد رفت.

این روش از سخت‌افزار حافظه مجازی جهت تشخیص کنترل انتقال‌ها به صفحات ایجادشده پویا استفاده می‌کند؛ و به‌طور خاص مجوزهای نوشتن و خواندن از صفحات حافظه مجازی را که هرکدام قابل‌اجرا یا قابل‌نوشتن باشد، دست‌کاری کرده و تضمین می‌کند که هرگز این دو مجوز باهم یکی نباشند. با استفاده از محافظت نوع نوشتن، این روش می‌تواند متوجه شود که صفحات تغییر یافته‌اند. با مجوز قابل‌اجرا نیز می‌تواند مشخص کند که کدام صفحات اجرا شده‌اند. اگر درجایی نیاز به تغییر مجوز صفات[1] صفحات شد، درحالی‌که این تغییرات در تضاد با تنظیمات ما باشد، این روش این‌که فایل چه قصدی از این عمل داشت را ثبت کرده ولی ما به‌صورت فیزیکی تنظیمات خود را نگه می‌داریم. اگر در ادامه همین فایل سؤالی مبنی بر استعلام مجوز داشت، باید همان پاسخی را داده شود که در مرحله قبلی صورت داده و از تنظیمات فیزیکی هیچ‌گونه اطلاعاتی داده نخواهد شد. هرگاه خطای محافظت حافظه مجازی رخ داد، کنترل را در دست گرفته و ابتدا بررسی می‌کند که آیا این خطا مربوط به تنظیمات خودش هست یا نه. اگر جواب منفی باشد، همان به‌طور ساده همان خطایی را که رخ‌داده به برنامه فراخواننده تحویل می‌شود، در غیر این‌صورت، صفات محافظتی را بر اساس روش بالا تغییر می‌دهد.

برای مطمئن شدن از اینکه برنامه اصلی در شکل بسته‌بندی‌شده‌اش در محیطی مشابه می‌تواند اجرا شود، بیشتر بازگشایی‌کننده‌ها پشته را خالی می‌کنند تا این‌که برنامه تعبیه‌شده بازگشایی شود و کنترل به آن منتقل شود، در این‌صورت پشته شبیه وقتی است که برنامه تعبیه‌شده به‌طور مستقیم درون حافظه بارگذاری شود. برای مثال فرض کنید در ابتدا ESP موقعی که برنامه بسته‌بندی‌شده شروع می‌شود برابر x500fc۰ باشد، سپس درست بعد از این‌که بازگشایی انجام شد، ESP باید دوباره به x500fc۰ اشاره کند. این قانون در مورد خیلی از بازگشایی‌کننده‌ها و حتی بازگشایی دستی به‌کار برده می‌شود. با ثبت مقدار ESP و مقایسه آن به هنگام هر بار اجرا در زمان انتقال کنترل به صفحه جدید ایجادشده، این روش، روشمند شده است. وقتی فایل اجرایی قابل‌حمل توسط خط فرمان و با استفاده از آرگومان‌ها اجرا می‌شود، این آرگومان‌ها ابتدا توسط بارگذاری‌کننده در حافظه heap قرار داده‌شده و سپس به‌وسیله تکه کد تولیدشده توسط کامپایلر، به هنگام بالا آمدن برنامه درون آن وارد می‌شود. بر این اساس کسی می‌تواند شروع اجرای فایل اصلی تعبیه‌شده درون فایل بسته‌بندی‌شده را تشخیص دهد که بعد از انتهای بازگشایی کوتاه رخ دهد.

اما عیب این روش در این است که با نگاشت دو حافظه مجازی بر روی صفحات فیزیکی مشابه که در یک صفحه تنظیمات غیرقابل نوشتن و قابل اجرا و در صفحه دیگر تنظیمات قابل نوشتن و غیرقابل اجرا انجام‌شده می‌توان به‌راحتی این روش را دور زد؛ که در این‌صورت بازگشایی‌کننده به‌راحتی می‌تواند در پس‌زمینه صفحات فیزیکی را از طریق صفحه مجازی قابل‌نوشتن و پرش به صفحه فیزیکی از طریق صفحه مجازی قابل‌اجرا بدون رخ دادن هیچ‌گونه خطایی صورت گیرد. البته در این روش برای جلوگیری از صدمه خوردن از این، مطمئن می‌شویم که صفات محافظتی که صفحات مجازی به صفحه فیزیکی نگاشت شده‌اند تنها از یک طریق صورت‌گرفته و در واقع از نگاشت چندگانه جلوگیری می‌کنیم.

به‌جای بخش PE، بازگشایی‌کننده می‌تواند خروجی بازگشایی را درون یک ناحیه حافظه پویا بگذارد. برای جلوگیری از فرار فایل‌ها از بازگشایی، صفحات را درون حافظه‌ای که قابلیت بزرگ شدن دارند، قرار می‌دهیم. به‌طور مشابه نیز وقتی فایلی به فضای آدرس یک فرایند نگاشت شد، ناحیه نگاشت شده نیاز به ردیابی دارد. بازگشایی‌کننده همچنین می‌تواند خروجی را درون یک فایل قرار دهد و بعدازآن فرایندی می‌تواند از آن فایل تکثیر شود. در این مورد این روش هیچ‌گونه اخطاری نخواهد داد، زیرا کد تولیدشده توسط یک فرایند ایجادشده درخواست شده و نه یک دستور پرش. البته این مورد توسط بررسی‌کننده بدافزار می‌تواند تشخیص داده شود ولی موقعی می‌تواند این کار صورت گیرد که فایل اجرا شده است.

[1] Properties

بعد از ایجاد دوباره فایل تعبیه‌شده، بعضی از بسته‌بندی‌کننده‌ها فرایند جدید را تکثیر داده و از درون فرایند جدید به فایل تعبیه‌شده پرش می‌کنند. این روش گریز به‌دلیل این‌که تمامی تنظیمات اعمال‌شده لزوماً به‌کل فرایندهای تکثیرشده از آن فرایند اعمال نشده، مؤثر است. البته این روش، این ترفند را نیز با ردگیری تنظیمات محافظ صفحه تمام فرایندهای تکثیرشده، مغلوب می‌کند.

بعضی از نندهها روش‌هایی را جهت مبارزه با فایل‌هایی که تلاش می‌کنند تا بفهمند در داخل نمونه‌ساز یا هرجایی که فعالیت‌هایش را رصد می‌کند، اجرا می‌شود را درون خودشان تعبیه می‌کنند. به‌دلیل این‌که این روش با روشی متفاوت از سایر بازگشایی‌کننده‌ها و با تغییر خصوصیات محافظتی صفحه استفاده می‌کند، گاهی اوقات با راه انداختن این روش‌های ضد نمونه‌سازی باعث توقف برنامه می‌شود. برای مثال اگر بازگشایی‌کننده‌ای صفحه قابل‌نوشتن با بافری که فرضاً به‌عنوان قابل نوشتن عبور می‌دهد را تشخیص دهد و این روش آن را درون هسته به‌عنوان آرگومان فراخوان سیستمی غیرقابل نوشتن علامت‌گذاری کند، وقتی هسته جهت نوشتن بافر تلاش می‌کند، خطای محافظ صفحه رخ خواهد داد و در نتیجه برنامه متوقف خواهد شد. این روش هرگز شانسی برای به‌دست آوردن نوع این خطا نخواهد داشت، زیرا این خطا خطای سطح هسته بوده و هرگز به سطح کاربر تحویل داده نخواهد شد. البته برای حل این مشکل سعی می‌شود از بروز خطای محافظ سطح هسته جلوگیری کرد و تغییرات خصوصیات محافظ صفحه اجازه می‌دهد که برنامه به اجرایش ادامه دهد و تغییرات قبل از برگشت به فراخوانی سیستمی، بازخواهند گشت.

وقتی انتهای بازگشایی تشخیص داده شد، به سراغ آدرس مقصد دستور انتقال کنترل که همان نقطه ورود برنامه[1] تعبیه‌شده درون فایل است، خواهیم رفت. ازآنجایی‌که بعضی از بسته‌بندی‌کننده‌ها نقطه ورود اصلی را با استفاده از جایگزین کردن چندین دستور به‌جای یک دستور پرش، مبهم می‌کنند، ناحیه‌ای به‌عنوان تکه کد مجزایی که شامل دستورات نقطه ورود اصلی و پرش به عقب با پیروی از این ناحیه مجزا، در نظر گرفته می‌شود. در این روش می‌توان به‌صورت تک‌مرحله‌ای چندین دستور ابتدایی را در نقطه ورود فرضی جایگزین کرد و به‌طور خاص این روش گریز را شناسایی کنیم. بعضی از بسته‌بندی‌کننده‌ها ورودی‌های دودویی بامعنی و مفهوم را در ابتدا و قبل از بسته‌بندی آن، اضافه می‌کنند. در حالت کلی این اضافه کردن‌ها همیشه صحیح و بی‌خطر نیستند، زیرا این کار نیاز به تجزیه کامل و مهندسی معکوس صحیح و با پوشش کامل دارد که به‌طور کامل امکان‌پذیر نیست؛ بنابراین، از آنجایی‌که این بسته‌بندی‌کننده‌ها ممکن است بعد از این‌که کار بازگشایی توسط این روش تمام شد، از دست موتور جستجوی بدافزار بر پایه امضاء، بگریزند، در حالت کلی این بسته‌بندی‌کننده‌ها زیاد قابل اتکا و اطمینان نبوده و خطرات و تهدیدات چندان جدی ندارند.

۳-۱- الگوریتم روش پیشنهادی

این الگوریتم تضمین می‌کند هر تکه کدی که در طول اجرای برنامه مشاهده شد، قبل از این‌که صدمات جبران‌ناپذیری به سیستم میزبان برساند، به‌وسیله تشخیص‌دهنده بدافزارش تشخیص داده شده و از بروز این صدمه جلوگیری کند. این الگوریتم روند اجرای برنامه را ردیابی کرده و به هنگام تولید کد جدید، تشخیص‌دهنده بدافزارش صدا زده می‌شود. برنامه در طول اجرای تشخیص‌دهنده بدافزار متوقف‌شده و اگر هیچ کد بدخواهی مشاهده نشد اجرا ادامه می‌یابد. بر این اساس وقتی یک مرحله بازگشایی تمام شد کد ایجادشده تازه، برای جستجوگر آماده خواهد شد.

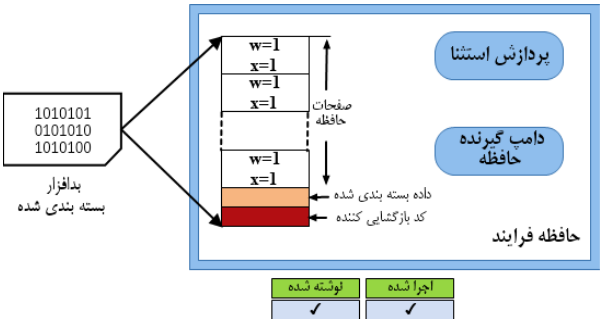
متأسفانه به‌دلیل نرخ بالای مراحل بازگشایی جعلی، فراخوانی تشخیص‌دهنده در پایان هر مرحله بازگشایی شاید هزینه زیادی داشته باشد، با تکیه بر این حقیقت که سیستم میزبان هیچ موقع بدون فراخوانی سیستمی خطرناک، با خطر مواجه نخواهد شد، می‌توان تشخیص‌دهنده بدافزار را تا زمان فراخوانی سیستمی به تعویق انداخت تا با این کار مقداری از سربار حاصله که در بالا ذکر شد کاهش یابد. مزیت به تعویق انداختن تشخیص‌دهنده بدافزار این است که خروجی چندین مرحله بازگشایی یک مرتبه و یکجا تحلیل می‌شود؛ که شبه کد آن در ادامه آورده شده است.

ابتدا فایل یا بدافزار پک شده بر روی حافظه بارگذاری می‌شود که در ابتدا صفات اولیه صفحات به‌صورت هم قابل نوشتن[2] و هم قابل اجرا شدن[3] است (شکل ۱). در مرحله اول تمام صفحات به‌صورت غیرقابل نوشتن ولی قابل اجرا شدن تنظیم می‌شود (شکل (۲)). سپس کد بازگشایی‌کننده وقتی که اقدام به نوشتن درون حافظه می‌کند، با خطای نوشتن در حافظه مواجه می‌شود (شکل (۳)) که در این مرحله این صفحه از حافظه به‌صورت غیرقابل اجرا و قابل نوشتن تنظیم می‌شود (شکل (۴)) و همین‌طور روند اجرای کد بازگشایی‌کننده ادامه می‌یابد که با صفحه بعدی نیز

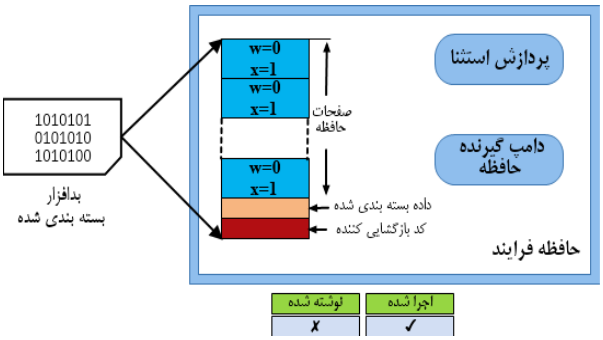
[1] OEP

[2] Writable
[3] Executble

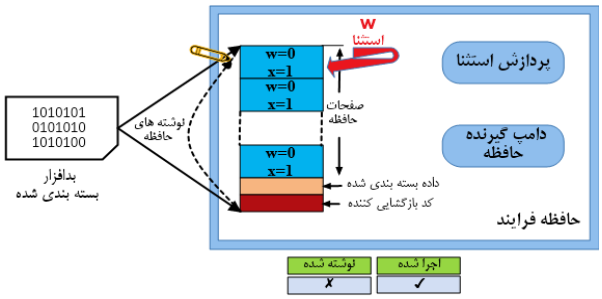
همین برخورد را خواهد داشت یعنی ابتدا با خطای نوشتن در حافظه مواجه می‌شویم (شکل (۵)).



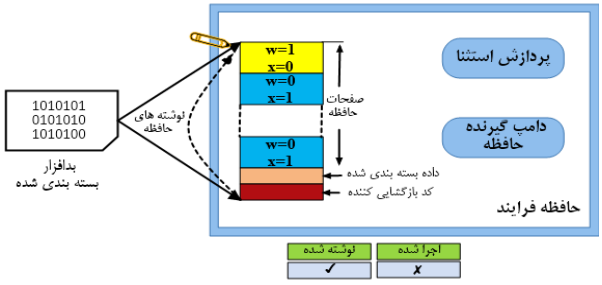
شکل (۱): حالت اولیه صفحات قابل‌نوشتن و اجراشدن



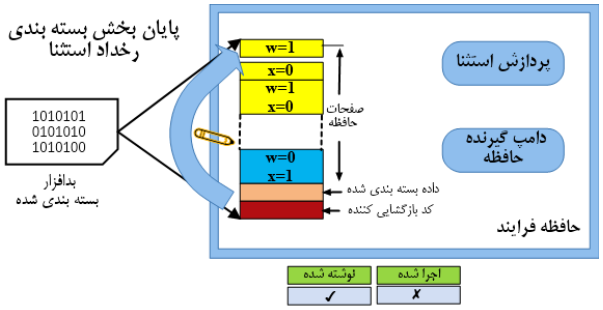
شکل (۲): تغییر صفحات قابل‌نوشتن به غیرقابل‌نوشتن



شکل (۳): خطای نوشتن به هنگام تلاش برای نوشتن

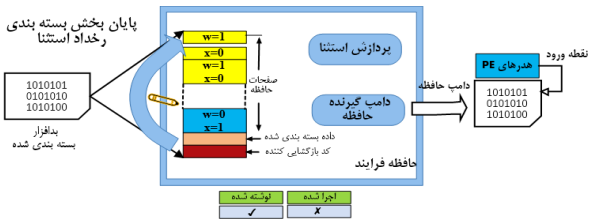


شکل (۴): تغییر صفحه به حالت غیرقابل‌نوشتن و اجراشدن



شکل (۵): خطای اجرای صفحه

این صفحه پیشتر به‌صورت غیرقابل نوشتن تنظیم شده بود که مجدد به‌صورت قابل نوشتن ولی غیرقابل اجرا تنظیم گردید. وقتی اجرای کد بازگشایی‌کننده به انتها رسید بازگشایی‌کننده می‌خواهد به نقطه‌ای پرش کند که قصد اجرا کردن کدهای آن صفحه را دارد و چون قبلاً صفحه غیرقابل اجراشدن گردید به خطای غیرقابل نوشتن برخورد می‌کند و در حقیقت اینجا همان نقطه‌ای است که کنترل‌کننده استثناء[1] فراخوانی می‌شود که بخش دامپ گیرنده حافظه[2] را فراخوانی کند تا از حالت فعلی فرایند که در حافظه است و دقیقاً در نقطه ورود به برنامه اصلی و بخش بازگشایی‌شده قرار دارد، دامپ بگیرد. سپس می‌تواند آن را به یک اسمبلر بفرستد تا تحلیل رفتاری یا هر نوع فعالیت دیگری روی آن صورت گیرد (شکل (۶)). در واقع با این کار به‌نوعی نقطه ورود برنامه به این صورت تغییر می‌یابد که نتیجه این فرایند، فایل اجرایی قابل حمل بازگشایی‌شده است یعنی حالتی از فایل که قبل از این‌که هرگونه بسته‌بندی بر روی آن انجام گیرد.



شکل (۶): انتقال به کنترل‌کننده خطا و گرفتن دامپ حافظه

با توجه به توضیحات ارائه‌شده، دو الگوریتم در شکل‌های (۷ و ۸) برای پیاده‌سازی مراحل ذکرشده ارائه می‌گردد.

1 W←0;/*written pages */

2 WX←0;/*pages written then executed*/

3 **foreach** $e \in \mathcal{I}$ **do**

4 **switch** *the value of e* **do**

5 **case** *write access w(p)*

6 W←$W \cup \{p\}$

7 **case** *instruction execution x(p)*

[1] Exception
[2] Memmory Dump

8 **if** $p \epsilon W$ **then**
9 $WX \leftarrow WX \cup \{p\}$
10 **case** *system-call invocation s*
11 **if** $s \epsilon \{dangerous\ system\ calls\} \wedge WX \neq 0$ **then**
12 **foreach** $p \epsilon W$ **do**
13 $r \leftarrow Scan(p)$
14 **if** $r = MALICIOUS$ **then**
15 halt execution
16 **return**

17 invoke the system-call handler for s
18 W←0
19 WX←0

شکل (۷): هسته اصلی الگوریتم روش پیشنهادی

1 **Input:** A packed binary program B
2 **Output:** a reconstructed PE file containing unpached program codes
3 **STEP 1:**
4 Load the packed program into memory
5 **for** *all p* in the program's memory pages **do**
6 *Permission(p)*|=***W*//recmovewritepermission**
7 **end for**
8
9 **STEP 2:**
10 **while** *B* is running **and** $T_{runtime} < T_{thresh}$ **do**
11 a: The address of the page fault
12 t: The page fault type t ϵ**{WRITE, EXECUTE}**
13 p←*page(a)*
14 **if t=WRITE then**
15 $permission(p) = (W|\bar{X})$*// Writble but non-executable*
16 *lost_writen(p)* ← current time
17 **end if**
18 **if** t=EXECUTE **then**
19 $permission(p) = (W|\bar{X})$*// non-Writble but executable*
20 *lost_exec(p)* ← current time
21 *addr-exec(k)* ← a
22 **end if**
23 **end while**
24
25 **STEP 3:**
26 Dump process memory
27 reconstruct *B'* by setting OEP to be *addr_exec(k)* where:
28 $k = \arg\min_k \text{lost\_exec(k)} > \max(\text{lost\_writen(i)})$
29 return *B'*

شکل (۸): الگوریتم روش پیشنهادی

در الگوریتم نشان داده‌شده در شکل (۸):

- W برای صفحات حافظه‌ای که نوشته شده‌اند، به‌کار گرفته شده است.
- WX برای صفحاتی که نوشته‌شده و سپس اجراشده‌اند، به‌کار گرفته شده است.
- e0 یا e1 یا ... برای ورودی الگوریتم که دنباله‌ای از اجراهاست، به‌کار گرفته شده است.
- رویداد ei توسط w(p) دسترسی نوشتن به حافظه را پیدا می‌کند.
- x(p) اجرای دستور از حافظه و درخواست فراخوانی سیستمی نیز با s است.

محرض هست که WX زیرمجموعه W است. وقتی برنامه بعد از اجرای کد نوشته‌شده قبلی در حافظه، فراخوانی سیستمی خطرناکی را صدا زد (خط ۱۱)، الگوریتم فوق، تشخیص‌دهنده بدافزار را برای تمام صفحات نوشته‌شده راه خواهد انداخت (خانه‌هایی که W شده‌اند)، اگر بدافزاری تشخیص داده شد، فرایند متوقف خواهد شد (خط ۱۴)، در غیر این صورت فراخوانی سیستمی اجازه اجرا پیداکرده و فرایند به کارش ادامه خواهد داد. بعدازاینکه کار بررسی تمام شد، مجموعه‌های W و WX دوباره تنظیم خواهند شد تا بار دیگر مورد بررسی قرار نگیرند (به‌جز آن‌هایی که یک‌بار قبل از اجرا تغییر داده شده‌اند).

در انتهای مرحله بازگشایی فقط صفحات نوشته‌شده- سپس اجراشده بررسی نمی‌شود، بلکه تمام صفحات نوشته‌شده بررسی می‌شود؛ بنابراین، اگر بازگشایی تمام شد دیگر نیازی به بررسی مجدد حافظه در طول باقیمانده اجرا نیست. به این نکته باید توجه کرد که این الگوریتم نیازی به مشاهده و بررسی دسترسی‌های تمام صفحات حافظه ندارد. تنها اولین دسترسی به حافظه‌ای که جهت دسترسی به هم نوع متوقف نشده، خواهد بود، برای مثال تنها اولین نوشته بر روی صفحه‌ای مفید خواهد بود که در ادامه بر روی همان صفحه نوشته می‌شوند؛ به‌عبارت‌دیگر تنها آن اطلاعاتی موردنیاز است که اولین اجرا بعد از دسترسی به همان صفحه برای نوشتن است و درنهایت این الگوریتم به‌دلیل این ویژگی‌ها کارایی بالا و سربار کمی خواهد داشت.

جهت بهینه‌کردن این فرایند و این‌که تشخیص نقطه ورود اصلی ناحیه بسته‌بندی‌شده آسان‌تر شود الگوریتم زیر اضافه می‌شود.

الگوریتم ارائه‌شده در شکل (۹)، بدین‌صورت کار می‌کند که تمام صفحاتی که با استفاده از ردیابی صفحات حافظه و کنترل سطح صفحه مشخص‌شده که صفت اجراشدن (ex) را دارند به حالت فقط خواندنی (r) تغییر می‌کند. با این تغییری که روی صفحات حافظه انجام داده شد، هر زمان که صفحه‌ای که نوشته‌شده و سپس قصد اجراشدن دارد، به‌دلیل این‌که مجوز دست‌یابی وجود ندارد و این صفحه فقط به‌صورت فقط خواندنی قابل‌دست‌یابی است و اعمال تغییر ممکن نیست این عمل باعث می‌شود الگوریتم به مرحله حالت استثنا[1] خواهد رفت و درواقع بخش مربوط به کنترل خطا اجرا خواهد شد که در این قسمت دقیقاً ما در نقطه‌ای قرار داریم که مکان شروع اجرای فایل اصلی است، دقیقاً همان نقطه اصلی ورودی که قبل از بسته‌بندی شدن است. وقتی نقطه اصلی ورود برنامه پیدا شد، به‌مثابه این است که فایل اصلی در اختیار است و روند و جریان اجرای برنامه تجزیه‌وتحلیل خواهد شد و اینجا وظیفه بخش موتور ضد بدافزار است که با آنالیز بخش‌های مختلف این بخش تشخیص دهد که آیا اجرای کدهای این بخش و درواقع به‌عبارت‌دیگر اجرای این فایل به‌سلامت سیستم آسیب می‌رساند یا نه.

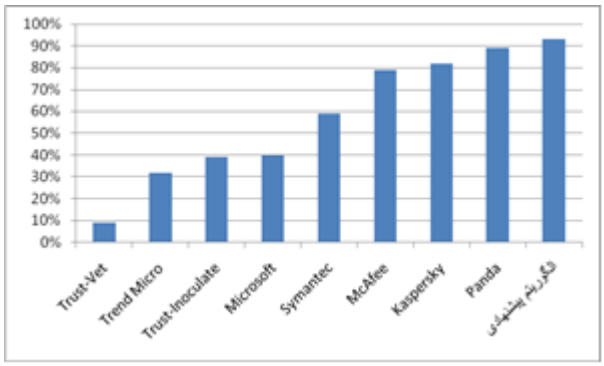
1 $W \leftarrow 0$;
2 $WX \leftarrow 0$;
3 Foreach e $\in \varphi$
4 **swhitch** the value of e **do**
5 **case** write access w(p)
6 $W \leftarrow W \cup \{p\}$
7 setReadonlyAttribute(p)

شکل (۹): الگوریتم فقط خواندنی کردن صفحات نوشته‌شده

۲- ارزیابی روش پیشنهادی

الگوریتم ارائه‌شده با کمک دو زبان C++[2] و زبان C# بر روی تعدادی از بدافزارهای موجود در virusshare و virussign مورداستفاده قرار گرفت تا کارایی و نحوه عملکرد آن مورد ارزیابی قرار گیرد...

برای انجام این آزمون حدود سیصد هزار بدافزار از دسته بدافزارهای ویروس، اسب‌های تروآ[3]، روت کیت[4]، بک دور[5]، جاسوس‌افزارها[6] و کی لاگرها[7] که هر دسته نزدیک شصت هزار بدافزار است دسته‌بندی شده‌اند. لازم به توضیح است که این عملیات بر روی یک ماشین باقدرت پردازشی دو پردازنده چهار هسته‌ای 2.4GH و حافظه اولیه 8G انجام شده است. همچنین نوع بسته‌بندی هر بدافزار را با استفاده از نرم‌افزار PEID و Packer Detector RDG به‌عنوان نرم‌افزارهای شناخته‌شده در این زمینه مورد استفاده قرار گرفتند. در شکل (۱۰) نتایج حاکی از آن است که اولاً ضد بدافزار[8]ها در بازگشایی فایل‌هایی که بسته‌بندی شده‌اند با مشکلاتی روبرو هستند که در واقع همان پیچیدگی روش‌های بسته‌بندی است و آن‌ها را از بازگشایی و سپس تجزیه‌وتحلیل فایل و در نتیجه از تشخیص بد افزار بودن آن عاجز هستند و روشی که پیشنهاد می‌شود در نمونه‌ای که انجام گردید و هرکدام از ابزارهای رسمی آن‌ها مورد تست قرار گرفت، روش پیشنهادی بیش از ۹۰ درصد از این فایل‌ها را با موفقیت بازگشایی نموده است که می‌تواند خوراک موتور ضد بدافزار را فراهم کند.



شکل (۱۰): بازگشایی موتور ضد بدافزارها برای فایل‌های بسته‌بندی‌شده

نتایج درصد بازگشایی در موتورهای بدافزار برای انواع بسته‌بندی کننده‌ها همراه با نمونه‌سازی و پیاده‌سازی الگوریتم پیشنهادی در شکل (۱) و جدول (۲) آمده است. همچنین در جدول (۱) نیز ارزیابی کارایی روش پیشنهادی جهت مهم‌ترین بخش بازگشایی یعنی پیدا کردن نقطه ورود اصلی یا همان OEP نشان داده‌شده است.

جدول (۱): ارزیابی تعداد شناخت موفق OEP[9]

بسته‌بندی کننده	تعداد فایل PE	تعداد OEP معتبر
Mpress	۲۰۰	۲۰۰
PeCompact	۳۰۰	۳۰۰
UPX	۷۰۰	۷۰۰
FSG	۱۵۰	۱۵۰
AsPack	۳۰۰	۳۰۰
PeTite	۱۰۰	۱۰۰
EZIP	۲۴۰	۲۴۰
NeoLite	۲۸۰	۲۸۰
SecuPack	۴۰۰	۴۰۰
cEXE	۱۲۰	۱۲۰

1- Exception
2- C++ Native
3- Trojan horse
4- Rootkit
5- backdoor
6- spyware
7- Key logger
8- Anti Virus
9- Original Entry Point

ارزیابی که برای برخی بسته‌بندی‌کننده‌ها جهت به‌دست آوردن کارایی این روش دریافتن OEP انجام شد (جدول ۱)، حاکی از آن است که این روش برای مهم‌ترین بخش بازگشایی یعنی یافتن نقطه انتهایی بخش بسته‌کننده و نقطه ابتدای بخش بازگشایی، بهترین عملکرد را دارد. این جدول نشان می‌دهد روش پیشنهادی در تمامی نمونه‌ها توانسته نقطه ورود اصلی فایل‌های بسته‌بندی‌شده را به‌درستی پیدا کند تا حساس‌ترین مرحله بازگشایی با موفقیت صورت گیرد. لازم به ذکر است که صحت عملکرد این ارزیابی با ابزارهایی مثل ollyDbg, IDAPro, UniversalUnpacker, PEid بررسی شد.

جدول (۲): نتایج درصد بازگشایی موتورهای بدافزار برای انواع بسته‌بندی‌کننده‌ها

	الگوریتم پیشنهادی	Panda	Kaspersky	McAfee	Symantec
میانگین	۹۴	۸۹	۷۲	۷۹	۵۹
ASpack	۹۸	۹۵	۹۷	۹۵	۸۱
FSG	۹۸	۱۰۰	۱۰۰	۵۶	۱۰۰
Morphine	۸۸	۱۰۰	۷۰	۱۰۰	۰
UPX	۱۰۰	۹۶	۹۷	۹۲	۱۰۰
MEW	۱۰۰	۱۰۰	۸۶	۵۳	۸۰
Armadillo	۷۸	۷۰	۱۴	۷۸	۲۶
Asprotect	۱۰۰	۸۰	۷۸	۸۵	۱۴
Obsidium	۷۳	۸۱	۵۵	۵۵	۲۷
Exe32pack	۱۰۰	۸۰	۳۳	۷۸	۳۵
PEBundle	۱۰۰	۸۹	۷۸	۸۰	۷۸
Yoda's	۹۳	۷۹	۷۵	۷۸	۷۸
Noelite	۱۰۰	۸۸	۸۰	۸۷	۵۵
PECompact	۱۰۰	۹۳	۶۷	۸۹	۸۸

همان‌طور که مشاهده می‌شود، نتایج ارزیابی از کارایی بالای این روش دارد و در مواردی هم که ملاحظه می‌شود درصدی کمتر از ۹۰ برای برخی بسته کننده‌ها وجود دارد، به دلیل استفاده آن نوع از بسته کننده‌ها از روش‌های مختلف و پیشرفته جهت ضد دیباگینگ و همچنین بهره بردن از نقاط ضعف این روش که قبلاً اشاره شد و یا حتی در هنگام ساخت IAT[1] یا همان جدول آدرس ورودی، است ولی در حالت کلی و به‌طور میانگین نسبت به سایر روش‌هایی که در موتورهای جستجوگر کنونی که در رده‌های بالایی از تشخیص‌دهنده‌های بدافزار قرار دارند، عملکرد بهتری حداقل در نتیجه دارد.

۵- نتیجه‌گیری

نتیجه‌ای که از این پژوهش حاصل می‌شود این است که دیگر با توجه به رشد روزافزون انواع بسته‌بندی کننده‌ها و پیچیدگی‌های هرکدام، عملاً روش‌های سنتی و دستی و بازگشایی یک‌به‌یک هرکدام امکان‌پذیر نبوده و باید به دنبال روش‌هایی همچون روش پیشنهادی در این مقاله بود و از ضررهای احتمالی جبران‌ناپذیر جلوگیری کرد.

به‌عنوان راه‌کار آتی نیز برای این مسئله می‌توان روش موجود را با موتور ضد بدافزارها ادغام نمود تا ضد بدافزار قوی‌تری داشته باشیم و همچنین نمونه‌ای از روش پیشنهادی را برای سایر سیستم‌عامل‌ها فراهم نمود، هرچند یافتن و ایجاد راه‌کاری که بتواند ۱۰۰% بسته‌بندی‌کننده‌ها و به‌خصوص بسته‌بندی‌کننده‌های آتی را نیز مورد پوشش خود قرار دهد چالشی بس بزرگ خواهد بود.

۶- منابع

1. A. Sharma and S. K. Sahay, "Evolution and Detection of Polymorphic and Metamorphic Malwares," International Journal of Computer Applications, vol. 90, no. 2, p. 7, June 2014.
2. W. Yang, A. Yuanyuan, Z. Juanru, L. Junliang S. Bodong, L. Wenjun, and H. Dawu Gu, "Bytecode Decrypting and DEX Reassembling for Packed Android Malware," Part of the Lecture Notes in Computer Science book series (LNCS, volume 9404), Dec. 2015.
3. S. Jain and Y. K. Meena, "Byte Level n–Gram Analysis for Malware Detection," In: K. R.Venugopal, L. M. Patnaik, (eds.) ICIP 2011. CCIS, vol. 157, pp. 51–59, Springer, Heidelberg, 2011.
4. T. Graf, "Generic unpacking: How to handle modified or unknown PE compression engines," Virus Bulletin, 2005.
5. L. Martignoni, Ch. Mihai, and J. Somesh, "Omniunpack: Fast, generic, and safe unpacking of malware," In Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual, pp. 431-441. IEEE, 2007.
6. N. Pors, "The Effectiveness of Self-Mutating Malware Detection Techniques," Utica College, ProQuest Dissertations Publishing, May 2017.
7. X. Tan, "Anti-unpacker tricks in malicious code," In Proceedings of 10th Annual AVAR International Conference, 2007.
8. M. Christodorescu, J. Somesh, K. Johannes, K. Stefan, and V. Helmut, "Software transformations to improve malware detection," Journal in Computer Virology 3, no. 4, pp. 253-265, 2007.
9. S. Huda, J. Abawajy, M. Alazab, M. Abdollalihian, R. Islam, and J. Yearwood, "Hybrids of support vector machine wrapper and filter based framework for malware detection," Future Generation Computer Systems, vol. 55, Issue C, pp. 376-390, Feb. 2016.
10. G. Pék, E. Author, Z. Lázár, Z. Várnagy, M. Félegyházi, and L. Buttyán, "A Posteriori Detection of Malicious Code Loading by Memory Paging Analysis," ESORICS 2016: Computer Security – ESORICS 2016, pp. 199-216, Sep. 2016.

1- Import Address Table

11. S. Najari and E. Terik, “Common features of destructive detection methods using learning techniques,” The 9th Symposium on Advances in Science and Technology (9thSASTech), Mashhad, Iran, Dec. 2014.
12. G. C. Kessler, “Anti-forensics and the digital investigator,” In Australian Digital Forensics Conference, p. 1, 2007.
13. Y. Park, “We can still crack you! general unpacking method for android packer (not root),” In: Black Hat Asia, 2015.
14. M. G. Kang, P. Pongsin, and Y. Heng, “Renovo: A hidden code extractor for packed executables,” In Proceedings of the 2007 ACM workshop on Recurring malcode, ACM, pp. 46-53, 2007.
15. D. Regalado, H. Shon, H. Allen, E. Chris, N. Jonathan, S. Branko, L. Ryan, and S. Stephen, “Gray Hat Hacking the Ethical Hacker's Handbook,” McGraw-Hill Education Group, 2015.
16. P. Battista, F. Mercaldo, and V. Nardone, A. Santone, and C. A. Visaggio, “Identification Malware Families with Model Checking,” Department of Engineering, University of Sannio, Benevento, Italy, sept. 2015.
17. P. Szor, “The art of computer virus research and defense, Pearson Education, 2005.
18. S. Jose, “Deep Packer Inspection: A Longitudinal Study of the Complexity of Run-Time Packers,” 2015 IEEE Symposium on Security and Privacy (SP) (2015). CA, USA, May 2015.
19. J. Rajendran, V. Vedula, T. Labs, Austin, and R. Karri, “Detecting malicious modifications of data in third-party intellectual property cores,” DAC '15 Proceedings of the 52nd Annual Design Automation Conference Article, no. 112, June 2015.
20. J. Caballero, U. Zurtuza, and J. Ricardo, “Detection of Intrusions and Malware, and Vulnerability Assessment,” 13th International Conference, DIMVA 2016, San Sebastián, Spain, July 7-8, 2016.
21. A. Arora, T. Stallings, R. Hasan, and G. Warner, “Malware Secrets: De-Obfuscating in the Cloud,” June 2017.

Improving Omniunpack Alghorithm in Generic Unpacking PE File with Page Monitoring

Y. Shakoori, S. Parsa*

Abstract

Analysts used file signature comparison to detect malware and analyze the behavior of the executable file in the past. To prevent signature examination, the authors of the new and advanced malware used obfuscation methods to hide information, of which packaging is the most important and the most efficient one. This method encrypts and compresses the code, without harming the behavior of the original executable file and the code is obscure until it is executed. The methods that are now used to unpack these files are often methods that are specially designed for each type of packer for that file. There are other methods, such as Renovo and OmniUnpack for reopening that are known as public reopens, and actually cover the weakness of previous approaches which is the need to know the type of packager, but their main problem is finding the original point. The main entry of the program is the end of the unpacking section. Our approach to fix this problem is a method that detects this point using tracking memory pages and monitoring pages, then executes it, and then dumps the memory for creating a new file that has been unpacked. Our method has two advantages: first, there is no need for knowledge of packaging type, and second it can also be used for packers that are created in the future. Finally, in the evaluation section, we have shown that this method has a very high performance for current packers and more than 90% of them can be unpacked with it, so it can be used on an antivirus engine.

Key Words: *Generic unpacking, Packing, Tracking page memory, Static analyze, Dynamic analyze, PE file*

* Iran University of Science and Technology (Parsa@iust.ac.ir)- Writer-in-Charge