نشریه علمی پدافند غیرعامل

سال دهم، شماره ۲، تابستان ۱۳۹۸، (پیاپی ۳۸): صص ۴۲-۳۳

ارائه روشی برای کشف روت‌کیت‌ها مبتنی‌بر درون‌بینی ماشین مجازی

سعید پارسا[1*]، فاطمه جمشیدی‌نیا[2]

تاریخ دریافت: ۱۳۹۵/۱۱/۲۴
تاریخ پذیرش: ۱۳۹۷/۰۷/۲۵

چکیده

روت‌کیت‌های سطح هسته، به‌دلیل رفتار پنهان‌کارانه خود، به تهدیدات امنیتی جدی تبدیل شده‌اند. اغلب روت‌کیت‌های سطح هسته، با قلاب‌اندازی اشاره‌گرهای تابع موجود در هسته سیستم‌عامل، جریان کنترل سیستم را تغییر داده و به اهداف پنهان‌کارانه خود دست می‌یابند. بررسی‌ها نشان می‌دهد اکثر روش‌های ضدروت‌کیتی که یکپارچگی اشاره‌گرهای تابع موجود در حافظه هسته سیستم را بررسی می‌کنند حافظه پویای هسته را که هدف حمله روت‌کیت‌های پیشرفته هستند، بررسی نمی‌کنند. از طرف دیگر روت‌کیت‌های سطح هسته قادر به دست‌کاری ساختارهای هسته سیستم‌عامل بوده و می‌توانند در کار نرم‌افزارهای ضد بدافزاری اختلال ایجاد کنند. بنابراین، ابزارهای کشف روت‌کیت پیشین، که در داخل ماشین میزبانی که آن را محافظت می‌کنند، اجرا می‌شوند، در برابر تغییر و دور زدن، آسیب‌پذیر هستند. بنابراین، در روش‌های اخیر کشف بدافزارها از روش‌های مبتنی بر نظارت ماشین مجازی در سطح ناظر ممتاز استفاده می‌شود که قادرند بدون دخالت بدافزارهای ماشین مجازی، وضعیت سیستم در حال اجرا را بررسی کنند. هدف از این پژوهش، ارائه روشی مبتنی بر درون‌بینی ماشین مجازی، به‌منظور کشف روت‌کیت‌هایی است که با استفاده از راه‌کار تغییر جریان کنترل سیستم سعی در مخفی نمودن خود و بدافزارهای جانبی‌شان در حافظه اصلی دارند. روش پیشنهادی سعی دارد با استفاده از درون‌بینی ماشین مجازی، اشاره‌گرهای تابع در نواحی حافظه هسته سیستم‌عامل که بیشترین هدف روت‌کیت‌ها هستند را استخراج کرده و در سطح ناظر ممتاز، یکپارچگی آن‌ها را بررسی کند. روش پیشنهادی با یک مجموعه از روت‌کیت‌های شناخته شده که از روش‌های پیشرفته قلاب‌اندازی استفاده می‌کنند، ارزیابی شده و قادر است همه آن‌ها را شناسایی کند.

کلیدواژه‌ها: روت‌کیت، درون‌بینی ماشین مجازی، قلاب‌اندازی، اشاره‌گرهای تابع

۱- دانشیار، دانشگاه علم صنعت ایران، دانشکده مهندسی کامپیوتر، (parsa@iust.ac.ir) - نویسنده مسئول
۲- کارشناس ارشد، دانشگاه علم صنعت ایران، دانشکده مهندسی کامپیوتر

۱- مقدمه

روت‌کیت‌ها بدافزارهایی هستند که توسط دست‌کاری ساختارهای داخلی سیستم، دید اصلاح شده‌ای از حالت سیستم به کاربر ارائه می‌دهند. روت‌کیت‌ها را به خودی خود نمی‌توان مخرب یا خطرناک دانست، بلکه قرار گرفتن آنها در کنار کرم‌ها یا ویروس‌ها یا نوع استفاده از آن‌هاست که به آنان ماهیتی خطرناک می‌بخشد. روت‌کیت‌ها با تزریق فایل‌های کتابخانه‌ای مخرب و قلاب‌اندازی اشاره‌گرهای تابع[1] سیستم‌عامل و اصلاح مستقیم ساختارهای هسته سیستم‌عامل به اهداف پنهان‌سازی دست می‌یابند. اشاره‌گرهای توابع هسته سیستم‌عامل بخشی از هسته سیستم هستند که بیشترین حمله را توسط روت‌کیت‌ها دارند. اشاره‌گرهای توابع اشاره به ناحیه‌ای از حافظه دارند که در آنجا دستورات یک تابع ذخیره شده است [۱]. بدافزارها، به‌ویژه روت کیت‌های سطح هسته، اغلب فضای هسته سیستم‌عامل را برای حمله هدف می‌گیرند و با دست‌کاری کد مؤلفه‌های هسته، ساختارهای داده شناخته شده از قبیل جدول توصیف‌گر سرویس سیستم[2] (SSDT)، جدول توصیف‌گر وقفه[3] (IDT) و جدول آدرس ورودی[4] (IAT) امکان اجرای کد مخرب را فراهم می‌کنند. به‌عبارت دیگر بدافزارهای از این نوع، قلاب‌اندازی‌هایی را نصب می‌کنند تا بتوانند کنترل سیستم مورد حمله را در اختیار بگیرند. توسعه‌دهندگان روت‌کیت، برای جلوگیری از کشف‌شدن، اشاره‌گرهای تابع در ساختارهای داده هسته را هدف می‌گیرند، مخصوصاً آن‌هایی که به‌صورت پویا از حافظه مخزن هسته[5] و حافظه هیپ[6] اختصاص داده می‌شوند. اصلاح اشاره‌گرهای تابع به‌صورت پنهانی و در سطح وسیعی است. تعداد زیادی اشاره‌گر تابع (اغلب آن‌ها دوره حیاتشان در سیستم طولانی است) در داخل هسته سیستم‌عامل ویندوز وجود دارد. زمانی‌که هسته سیستم‌عامل مورد حمله قرار می‌گیرد، حمله‌کننده‌ها هر جنبه از سیستم قربانی را کنترل می‌کنند، آن‌ها می‌توانند عملکردهای نامشروع را به‌طور مستقیم پیاده‌سازی کنند، مؤلفه‌های مخرب سطح کاربر را پنهان کنند و کشف خود و مؤلفه‌هایشان را دشوار کنند. برای دست‌یابی به این اهداف مخرب، بدافزارها تمایل به ایجاد تغییرات جریان کنترل ماندگار دارند یا به عبارت دیگر قلاب‌اندازی‌های دائمی را در سیستم قربانی نصب می‌کنند. بدافزارهای پیشین، قلاب‌اندازی‌ها را در ناحیه کد هسته و ساختارهای داده شناخته‌شده انجام می‌دادند و ابزارهای کشف قلاب‌اندازی، یکپارچگی اشاره‌گرهای تابع را در این نواحی بررسی می‌کنند بنابراین، قادر به کشف قلاب‌اندازی‌ها در ساختارهای داده هسته که در نواحی پویا مانند حافظه مخزن هسته اختصاص داده می‌شوند، نیستند [۲]. این ساختارهای داده، تنظیمات و حالت‌های بحرانی سیستم را نگهداری می‌کنند و شامل اشاره‌گرهای تابع مهمی هستند. تعداد اشاره‌گرهای تابع در هسته می‌تواند زیاد باشد و بدون دانش عمیق از ساختارهای داده هسته، پیدا کردن آن‌ها مشکل است. بنابراین، این روش قلاب‌اندازی جدید برای نصب قلاب‌اندازی‌های پنهانی ایده‌آل است [۳]. از سوی دیگر، روش‌های کشف بدافزار موجود، برای شناسایی بدافزارها به اطلاعات ارائه‌شده توسط سیستم‌عامل وابسته هستند، بنابراین، نمی‌توانند روت‌کیت‌های سطح هسته را شناسایی کنند، زیرا این دسته از روت‌کیت‌ها قادر به دست‌کاری ساختارهای هسته سیستم‌عامل بوده و می‌توانند در کار نرم‌افزارهای ضد بدافزاری اختلال ایجاد کنند [۴]. تحقیقات زیادی برای حل این مشکل انجام شده است که در میان آن‌ها، فنّاوری مجازی‌سازی به موضوع تحقیقاتی مهم و مطرحی تبدیل شده است که بر محدودیت‌های اساسی روش‌های قدیمی مبتنی بر میزبان غلبه کرده است. دلیل اصلی انتخاب مجازی‌سازی برای کشف روت‌کیت‌ها این است که مجازی‌سازی قادر است ابزارهای امنیتی را از ماشین مجازی غیرقابل اعتماد جدا کند و به آن‌ها حق امتیاز بالاتری را برای نظارت ماشین مجازی غیرقابل اعتماد ارائه دهد. بنابراین، در روش‌های اخیر کشف بدافزارها از روش‌های مبتنی بر نظارت ماشین مجازی در سطح ناظر ممتاز برای کشف روت‌کیت‌ها استفاده می‌شود که قادرند بدون دخالت بدافزارهای ماشین مجازی، وضعیت سیستم در حال اجرا را بررسی کنند. به‌طور کلی هریک از روش‌های ارائه‌شده برای کشف قلاب‌اندازی‌های هسته سیستم‌عامل توسط روت‌کیت‌ها، حداقل دارای یکی از این محدودیت‌های است؛ وابسته به کد منبع هسته سیستم‌عامل، اجرا در سطح امتیاز یکسان با بدافزار، سربار عملکردی قابل توجه و عدم بررسی فضای پویای هسته [۵]. برای غلبه بر محدودیت‌های مطرح‌شده، در این پژوهش روشی ارائه شده است که یکپارچگی اشاره‌گرهای تابع ماندگار موجود در حافظه هسته سیستم‌عامل را بررسی می‌کند. این اشاره‌گرهای تابع، از ساختارهای شناخته‌شده سیستم‌عامل و ساختارهای داده موجود در فضای مخزن هسته استخراج می‌شوند. سپس در سطح ناظر ممتاز با استفاده از درون‌بینی ماشین مجازی به‌صورت دوره‌ای بررسی می‌شوند، تغییرات صورت گرفته در آن‌ها اغلب نشانه وجود روت‌کیت‌ها است. روش پیشنهادی با یک مجموعه از روت‌کیت‌های شناخته‌شده که از روش‌های پیشرفته قلاب‌اندازی

[1] Function Pointer Hooking
[2] System Service Descriptor Table
[3] Interrupt Descriptor Table
[4] Import Address Table
[5] Kernel Pool Memory
[6] Heap

استفاده می‌کنند، ارزیابی شده و قادر است همه آن‌ها را شناسایی کند.

در ادامه، در بخش دوم از مقاله، کارهای مرتبط انجام‌شده در زمینه کشف روت‌کیت‌ها شرح داده شده، در بخش سوم روش پیشنهادی ارائه شده، در بخش چهارم ارزیابی‌های انجام شده تشریح شده‌اند و در پایان نتیجه‌گیری ارائه شده است.

۲- کارهای مرتبط

بدافزارهای قدیمی، قلاب‌اندازی‌هایی را در نواحی کد هسته یا ساختارهای داده شناخته شده هسته نصب می‌کردند. این نواحی شامل جدول توصیف‌گر سرویس سیستم (SSDT)، جدول آدرس ورودی (IAT) و جدول توصیف‌گر وقفه (IDT) هستند. ابزارهای کشف قلاب‌اندازی موجود از قبیل VICE [۶] و IceSword [۷] یکپارچگی نواحی هسته و ساختارهای داده شناخته شده هسته را بررسی می‌کنند. اما بدافزارهای امروزی برای فرار از کشف شدن، قلاب‌اندازی‌ها را در نواحی حافظه مخزن هسته که به‌صورت پویا اختصاص داده می‌شوند انجام می‌دهند. بنابراین، نیاز به روشی است که بتواند یکپارچگی اشاره‌گرهای تابع در این نواحی را بررسی کند. روش‌های بسیاری برای مقابله با تغییر جریان کنترل هسته توسط روت‌کیت‌ها مطرح شده است. روش‌های ارائه‌شده در [۸-۱۱] یکپارچگی اشاره‌گرهای تابع را توسط ایجاد سیاست کشف قلاب‌اندازی و استخراج اطلاعات درباره اشاره‌گرهای تابع را با اجرای تحلیل ایستای کد منبع هسته سیستم‌عامل بررسی می‌کنند. برای مثال، این روش‌ها به تعریف ساختارهای داده هسته که شامل اشاره‌گرهای تابع هستند دست می‌یابند و متعاقباً گراف پیمایشی را ایجاد می‌کنند تا توسط آن به ساختارهای داده شامل اشاره‌گرهای تابع دست یابند. اما متأسفانه این راه‌حل‌ها وابسته به دسترس بودن کد منبع هسته هستند بنابراین، برای سیستم‌عامل‌های متن بسته مانند ویندوز مناسب نیستند.

در مقاله [۲]، ابزاری با نام هوک اسکات برای بررسی یکپارچگی اشاره‌گرهای تابع در سیستم‌عامل ویندوز ارائه شده است. این روش با ردیابی اشاره‌گرهای تابع در حافظه، یک سیاست کشف قلاب‌اندازی را با نظارت اشیاء حافظه دربرگیرنده اشاره‌گرهای تابع ایجاد می‌کند. ابزار ارائه شده اطلاعاتی را درباره اشاره‌گرهای تابع از یک سیستم‌عامل سالم که برنامه‌های کاربردی عادی روی آن نصب شده است، یاد می‌گیرد. موفقیت این روش بسته به این است که چه مقدار اطلاعات درباره اشاره‌گرهای تابع در مرحله یادگیری در دسترس است. در طول مرحله کشف، اگر ماشین هدف توسط دست‌کاری و تغییر اشاره‌گرهای تابع که در طول مرحله یادگیری مورد ارزیابی قرار نگرفتند، مورد حمله قرار گیرد قادر به بررسی یکپارچگی کامل اشاره‌گرهای تابع نخواهد نبود. ابزار ارائه شده برای کشف قلاب‌اندازی یک دستورالعمل پرش در نقطه ورود هر تابع که اشاره‌گر آن نظارت می‌شود، قرار می‌دهد. این دستور پرش، به کد کشف مربوطه هدایت می‌کند. این کار در سیستم‌عامل ویندوز ۶۴ به علت وجود پچ‌گارد غیرممکن است. هم‌چنین این روش به‌صورت یک مخلفه سطح هسته پیاده‌سازی شده است که در فضای هسته ماشینی که نظارت می‌شود اجرا می‌شود بنابراین، می‌تواند توسط بدافزارهای سیستم شناسایی و غیرفعال گردد. علاوه بر هوک اسکات، روش‌های دیگری برای مقابله با تغییر کنترل جریان هسته توسعه داده شده‌اند که بسیاری از آن‌ها نواحی پویای حافظه را که شامل اشاره‌گرهای تابع بسیاری هستند و هدف مطلوبی برای روت‌کیت‌ها محسوب می‌شوند را پوشش نداده و بنابراین، در بررسی یکپارچگی اشاره‌گرهای سطح هسته موفق نبوده‌اند. در ادامه روش‌های شرح داده می‌شوند که بر این مشکل غلبه کرده‌اند اما به علت محدودیت‌هایی از قبیل دسترس‌پذیری کد هسته و یا عملکرد غیرقابل قبول موفق نبوده‌اند. در مقاله [۱۲] روشی ارائه شده است که متکی به تحلیل ایستای کد منبع هسته یا نمادهای اشکال‌زدایی و اطلاعات نوع برای تحلیل دودویی است. علاوه بر این مؤلفه‌های اصلی روش ارائه‌شده داخل ماشین مجازی مورد نظارت اجرا می‌شوند و بنابراین، در معرض دست‌کاری بدافزارها قرار دارند و نمی‌توان به آن‌ها اعتماد کرد. به‌علاوه این روش، به‌دلیل بهره‌گیری از منابع اضافه دارای سربار قابل توجهی است. روشی در [۱۳] ارائه شده است که یک بررسی یکپارچگی سیستماتیک هسته سیستم‌عامل به‌وسیله نگاشت اشیاء هسته را اجرا می‌کند. این سیستم به‌صورت آفلاین عمل کرده و با گرفتن دامپ‌های حافظه هسته ویندوز و استفاده از دیباگر ویندوز به اطلاعات اشیاء هسته دست می‌یابد. در مقاله [۱۴] روشی ارائه شده است که یک کپی از اشاره‌گرهای تابع ثابت ایجاد می‌کند و آدرس نگاشت شده آن‌ها را به یک سخت‌افزار محافظ صفحه برمی‌گرداند. تقاضاها برای تغییر این کپی بازبینی می‌شود و به مؤلفه ناظر ممتاز فرستاده می‌شود تا صحت این تقاضاها بررسی شود. این روش نیاز به دانش کد منبع سیستم‌عامل دارد که همیشه در دسترس نیست. در [۳] روشی ارائه شده است که به‌صورت دوره‌ای یکپارچگی جریان کنترل هسته را بررسی می‌کند. این روش هم نیازمند دانش کد هسته و فایل‌های باینری است و همان‌طور که قبلاً ذکر شد کد منبع هسته برای سیستم‌عامل‌های متن بسته مانند ویندوز در دسترس نیست. روش ارائه شده در [۴] متکی به یک ماشین مشاهده کننده مجزا برای گرفتن تصاویر لحظه‌ای حافظه به‌صورت دوره‌ای از طریق دست‌یابی مستقیم حافظه برای تحلیل یکپارچگی است.

به‌دلیل استفاده از روش DMA سربار عملکردی ماشین مشاهده‌کننده قابل توجه است. به‌علاوه این روش به سخت‌افزار اضافه برای نظارت نیاز دارد و هم‌چنین نیاز به کد منبع سیستم‌عامل برای تحلیل ایستای اولیه دارد که همان‌طور که ذکر شد برای سیستم‌عامل متن بسته مانند ویندوز در دسترس نیست.

روشی برای کشف قلاب‌اندازی در سیستم‌عامل لینوکس در [۱۰] ارائه شده است که در سطح ناظر ممتاز با استفاده از نظارت حافظه، روت‌کیت‌ها را شناسایی می‌کند. روش کار به این صورت است که از آنجا که جدول فراخوانی سیستمی در نسخه‌های متعدد هسته لینوکس، تغییری نمی‌کند بنابراین، بر اساس یک الگوی ثابت در آدرس فراخوانی‌های سیستمی می‌توان، ورودی‌های این جدول را استخراج کرد. پس از یافتن فراخوانی‌های سیستمی، مقدار هش آدرس هریک از آن‌ها به همراه محتوای آن یعنی تابع مورد نظر در یک بانک اطلاعاتی ذخیره می‌شود. سپس موقع بررسی برای وجود روت‌کیت‌ها، دوباره این مقدار محاسبه شده و با مقدار ذخیره شده قبلی مقایسه می‌شود. عدم تطابق به معنی وجود تغییر در فراخوانی‌های سیستمی توسط روت‌کیت‌ها است. این روش با سربار کمی می‌تواند روت‌کیت‌ها را شناسایی کند اما از آنجا که الگوی آدرس موردنظر قابل اعمال به ویندوز نیست نمی‌توان از این روش برای سیستم‌عامل ویندوز استفاده کرد. از طرف دیگر این روش، محدود به جستجوی آثار روت‌کیت‌ها فقط در جدول فراخوانی‌های سیستمی است و نواحی مهم هسته سیستم‌عامل را بررسی نمی‌کند. روشی مبتنی بر درون‌بینی ماشین مجازی برای کشف روت‌کیت‌ها در [۱] ارائه شده است که در آن نواحی مهم حافظه که بیشتر هدف حمله روت‌کیت‌ها قرار دارند از قبیل جداول آدرس ورودی/خروجی مؤلفه‌های سطح هسته، جدول توصیف‌گر وقفه سیستم، جدول توصیف‌گر سرویس سیستم و ساختارهای داده سطح هسته، در سطح ناظر ممتاز محافظت می‌شود، هرگاه برنامه‌ای سعی کند این نواحی را دست‌کاری کند، مانع آن می‌شود. اما این روش نواحی حافظه را که به‌صورت پویا تخصیص داده می‌شوند و می‌توانند شامل داده‌های مهم از قبیل اشاره‌گرهای تابع باشند که هدف عمده روت‌کیت‌ها هستند را پوشش نداده و محافظت نمی‌کند.

در [۱۱] ابزاری با نام blacksheep ارائه شده است که از یک روش ساده برای شناسایی تغییر اشاره‌گرهای تابع استفاده می‌کند. این روش به‌طور دقیق محل اشاره‌گرهای تابع را شناسایی نمی‌کند، در عوض به‌طور مستقیم آن‌ها را از طریق یک روش تطابق الگو بر مبنای اختلاف منظر در میان تعدادی از ماشین‌های مجازی همگن شناسایی می‌کند. مشکل مطرح در این روش این است که سربار زیادی دارد و هم‌چنین ابزار تشخیص روت‌کیت در ماشین هدف اجرا می‌شود. روش ارئه‌شده در [۱۲] از تحلیل پویای کد هسته برای شناسایی و تحلیل قلاب‌اندازی‌ها استفاده می‌کند. این روش نظارت مداوم برای بررسی فعالیت روت‌کیت‌ها ندارد و به شدت منبع‌گرا است بنابراین، به‌طور بالقوه با نظارت عملکرد توسط بدافزارها قابل شناسایی است. ابزاری با نام Hooklocator در [۱۳] ارائه شده است که مبنای کار این پژوهش قرار گرفته است. این روش، یکپارچگی اشاره‌گرهای تابع فضای حافظه مخزن هسته را که به‌صورت پویا تخصیص داده شده‌اند را بررسی می‌کند. این روش در قالب ۴ مرحله مختلف انجام می‌شود در مرحله اول مؤلفه استخراج یک لیست از اشاره‌گرهای تابع از منابع قابل اعتماد در حافظه فیزیکی ماشین مجازی ایجاد می‌کند که این لیست توسط مؤلفه جستجو برای پیدا کردن اشاره‌گرهای تابع کاندید در داده حافظه مخزن هسته استفاده می‌شود. در مرحله دوم، ابتدا تخصیص‌های پویای هسته سیستم‌عامل استخراج شده و سپس این نواحی بر اساس لیست اشاره‌گرهای تابع مرحله قبل جستجو می‌شوند تا اشاره‌گرهای تابع موجود در آن‌ها استخراج شود. سپس مؤلفه یادگیری از اکتشاف برای شناسایی اشاره‌گرهای تابع اصل استفاده می‌کند و در نهایت، این اشاره‌گرها توسط مؤلفه نظارت برای بررسی یکپارچگی، نظارت می‌شوند. اما از آنجا که فضای مخزن هسته می‌تواند بزرگ باشد پویش کل آن بسیار زمان‌بر است. در روش پیشنهادی در این پژوهش این فضای جستجو کاهش داده می‌شود و فقط با بررسی نواحی که بیشترین اشاره‌گرهای تابع را دارند و هدف مطلوب روت‌کیت‌ها هستند زمان پویش حافظه بهبود داده می‌شود و هم‌چنین این روش، اشاره‌گرهای تابع در نواحی ثابت هسته سیستم‌عامل را نظارت نمی‌کند و روت‌کیت‌های جدید می‌توانند با دور زدن پچ‌گارد[1] ویندوز هفت ۶۴ این منابع را دست‌کاری کنند [۱۸]. هدف این پژوهش، حل این مشکلات و ارائه روشی است که علاوه بر پوشش کامل مهم‌ترین اشاره‌گرهای تابع هسته که هدف عمده روت‌کیت‌ها هستند، در سطح ناظر ممتاز اجرا شده و در معرض دست‌کاری روت‌کیت‌ها قرار نداشته باشد. به‌طور کلی هریک از روش‌های ذکر شده، حداقل دارای یکی از این محدودیت‌های است؛ وابسته به کد منبع هسته سیستم‌عامل، اجرا در سطح امتیاز یکسان با بدافزار، سربار عملکردی قابل توجه و عدم بررسی فضای پویای هسته. در این پژوهش سعی شده، روشی ارائه شود تا بر اغلب این محدودیت‌ها غلبه کند.

روش پیشنهادی، اشاره‌گرهای تابع موجود در منابع سیستم‌عامل از قبیل کد مؤلفه‌های هسته و ساختارهای داده

[1] PachGuard

سطح هسته و هم‌چنین فضای مخزن هسته سیستم‌عامل را در سطح ناظر ممتاز با استفاده از درون‌بینی ماشین مجازی استخراج کرده و به‌صورت مداوم آن‌ها را نظارت می‌کند تا تغییرات غیرمجاز انجام‌شده در آن‌ها توسط روت‌کیت‌ها را شناسایی کند. در روش پیشنهادی برای استخراج اشاره‌گرهای تابع موجود در فضای مخزن هسته از روش ارائه‌شده در [۱۳] استفاده شده است.

۳- روش پیشنهادی

روش پیشنهادی شامل سه مرحله است، مرحله اول، استخراج اشاره‌گرهای تابع از منابع سیستم‌عامل است، در این مرحله، اشاره‌گرهای تابع موجود در کد مؤلفه‌های هسته سیستم‌عامل، جداول آدرس ورودی/خروجی، جدول توصیف‌گر سرویس سیستم و جدول توصیف‌گر وقفه استخراج می‌شوند. در مرحله بعد، اشاره‌گرهای موجود در نواحی حافظه مخزن هسته بر طبق لیست اشاره‌گرهای مرحله قبل استخراج می‌شوند و در نهایت، در مرحله سوم، تمامی اشاره‌گرهای استخراج‌شده موجود در هسته به‌صورت دوره‌ای نظارت می‌شوند. در ادامه هریک از این مراحل به‌طور کامل تشریح شده‌اند.

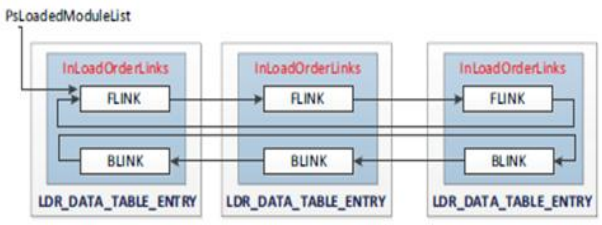
۳-۱- استخراج اشاره‌گرهای تابع از منابع سیستم عامل

در این مرحله، یک لیست از اشاره‌گرهای تابع موجود در کد مؤلفه‌های هسته و ساختارهای شناخته شده مقیم در حافظه فیزیکی ماشین مجازی میهمان، شامل جدول توصیف‌گر وقفه (IDT)، جدول توصیف‌گر سرویس‌های سیستم (SSDT)، جدول آدرس ورودی (IAT) و جدول آدرس خروجی (EAT) ایجاد می‌شود. ساختارهای داده در حافظه به‌خوبی سازماندهی شده‌اند و فیلدهای مشخصی دارند که می‌توان اشاره‌گرهای تابع را از آن‌ها استخراج کرد. اما کد مؤلفه‌های هسته شامل چنین فیلدهایی نبوده و برای استخراج اشاره‌گرهای تابع از آن نیازمند روش دیگری است. روش استفاده شده برای استخراج اشاره‌گرهای تابع از کد هسته، مقایسه متقابل[1] است که از مزیت‌های ویژگی جابجاپذیری[2] کد هسته ویندوز بهره می‌برد. برای شناسایی آدرس‌های مطلق (اشاره‌گرهای تابع) در کد مؤلفه‌های هسته باید دو تصویر از هر کد مؤلفه هسته بار شده در مکان‌های مختلف حافظه را مقایسه کرد. تفاوت در محتوای دو مؤلفه بارشده، آدرس‌های مطلق را شناسایی می‌کند. زیرا هسته در مکان‌های مختلف بار می‌شود و از طرفی خود کد هسته تغییری نمی‌کند و ثابت است. بنابراین، با شناسایی آدرس‌های مطلق در کد، می‌توان اشاره‌گرهای تابع موجود در کد را شناسایی کرد [۱۳]. این فرایند در شکل (۱) برای مؤلفه ntoskrnl هسته نمایش داده شده است. برای استخراج اطلاعات درباره اشاره‌گرهای تابع موجود در کد و جدول‌های آدرس ورودی و خروجی مؤلفه‌های هسته باید ابتدا اطلاعاتی درباره مؤلفه‌های هسته داخل حافظه از قبیل نام مؤلفه، آدرس پایه مؤلفه و اندازه آن جمع‌آوری کرد. ویندوز یک لیست از مؤلفه‌های هسته، در یک لیست پیوندی دو طرفه نگهداری می‌کند. هر گره در لیست با ساختار داده LDR_DATA_TABLE_ENTRY نمایش داده می‌شود. آدرس پایه اولین گره در لیست در متغیر سراسری سیستمی، PsLoadModuleList ذخیره شده است که برای دست‌یابی به این لیست استفاده می‌شود. برای استخراج اطلاعات مؤلفه‌های هسته از این لیست استفاده می‌شود. نمایی از این لیست در شکل (۲) نمایش داده شده است [۱۳].

```
000000000| bc 44 2b 86 76 44 2b 86  f6 44 2b 86 44 45 2b 86  .D+.vD+..D+.DE+.
000000010| 7c 45 2b 86 f2 43 2b 86  64 44 2b 86 2c 44 2b 86  |E+..C+.dD+.,D+.
000000020| ba 49 2b 86 24 b0 2b 86  bc 43 2b 86 62 48 2b 86  .I+.$.+..C+.bH+.
                        . . . .
0006c710| 30 01 83 f8 20 77 34 83  c1 04 3b 4d fc 72 eb 89  0... w4...;M.r..
0006c720| 55 10 ff 15 64 20 60 82  64 8b 0d 20 00 00 00 88  U...d `.d.. ....
                        . . . .
0011b8e0| 4e 8b 30 5b 46 d6 59 41  b8 71 17 c7 11 e7 34 0c  N.0[F.YA.q....4.
0011b8f0| 02 00 00 00 6e 74 6b 72  70 61 6d 70 2e 70 64 62  ....ntkrpamp.pdb
```

```
000000000| bc 24 28 86 76 24 28 86  f6 24 28 86 44 25 28 86  .$(.v$(..$(.D%(.
000000010| 7c 25 28 86 f2 23 28 86  64 24 28 86 2c 24 28 86  |%(..#(.d$(.,$(.
000000020| ba 29 28 86 24 90 28 86  bc 23 28 86 62 28 28 86  .)($.(..#(.b((.
                        . . . .
0006c710| 30 01 83 f8 20 77 34 83  c1 04 3b 4d fc 72 eb 89  0... w4...;M.r..
0006c720| 55 10 ff 15 64 00 65 82  64 8b 0d 20 00 00 00 88  U...d.e.d.. ....
                        . . . .
0011b8e0| 4e 8b 30 5b 46 d6 59 41  b8 71 17 c7 11 e7 34 0c  N.0[F.YA.q....4.
0011b8f0| 02 00 00 00 6e 74 6b 72  70 61 6d 70 2e 70 64 62  ....ntkrpamp.pdb
```

شکل (۱): آدرس‌های مطلق در دو تصویر متفاوت از کد مؤلفه هسته ntoskrnl



شکل (۲): لیست پیوند نگهداری مؤلفه‌های هسته

هریک از مؤلفه‌های موجود در لیست را می‌توان با استفاده از آدرس پایه[3] هر مؤلفه و اندازه آن از حافظه استخراج کرد. هر مؤلفه هسته در فرمت اجرایی قابل حمل[4] (PE) است که شامل قسمت‌های سرآمدها[5]، کد و داده و جداول آدرس ورودی و خروجی است. در این مرحله باید هر مؤلفه را در فرمت PE برای دست‌یابی به جداول و کد، تجزیه و تحلیل کرد. سپس اشاره

[1] Cross-comparision
[2] Relocatable
[3] Base Address
[4] Portable Executable
[5] Header

گرهای تابع داخل کد از روش ذکرشده مقایسه متقابل در بخش قبل به‌دست می‌آید و اشاره‌گرهای تابع مربوط به جدول‌های ورودی و خروجی هم از سرآمدهای مربوطه استخراج می‌شوند. برای استخراج اشاره‌گرهای موجود در مؤلفه‌های هسته با روش مقایسه متقابل ابتدا باید دو تصویر لحظه‌ای از هر مؤلفه، در بارگذاری‌های متفاوت سیستم‌عامل ایجاد شوند. سپس با استخراج قسمت متنی مولفه‌ها از فرمت PE و مقایسه بایت‌های آن‌ها، آدرس‌های مطلق[1] شناسایی و در نتیجه اشاره‌گرهای تابع استخراج می‌شوند [۱۵].

پس از استخراج اشاره‌گرهای تابع کد مؤلفه‌های هسته، باید اشاره‌گرهای تابع موجود در جدول توصیف‌گر سرویس سیستم (SSDT) استخراج شوند. جدول توصیف‌گر سرویس سیستم، شامل اشاره‌گرهایی به توابع سطح هسته است. جدول توصیف‌گر سرویس سیستم، هدف بسیاری از روت‌کیت‌ها برای پنهان‌سازی اشیاء موجود در حافظه است. برای مثال برخی از روت‌کیت‌ها با قلاب‌اندازی تابع ZwQuerySystemInformation فرآیندهای موردنظر خود را پنهان می‌کنند. برای دست‌یابی به این جدول از چارچوب ولاتیلیتی[2] استفاده شده است و سپس لیست ورودی‌های این جدول که همان اشاره‌گرهای تابع هستند استخراج می‌شوند. پلاگین SSDT در این ابزار، لیست ورودی‌های جدول همراه با نام هر تابع و آدرس آن در مؤلفه ntoskrnl را استخراج می‌کند.

جدول توصیف‌گر وقفه (IDT)، توسط پردازنده برای انتقال اجرا از برنامه در حال اجرا به روتین نرم‌افزاری خاص برای مدیریت وقفه‌ها استفاده می‌شود. حمله‌کننده‌ها برای اجرای کد مخرب خود مکرراً اشاره‌گرهای توابع این جدول را دست‌کاری می‌کنند. بنابراین، برای بررسی یکپارچگی اشاره‌گرهای توابع هسته لازم است یکپارچگی IDT نیز بررسی شود. برای دست‌یابی به این جدول از رجیستر IDTR استفاده می‌شود. با استفاده از این رجیستر می‌توان به آدرس پایه و اندازه جدول دست یافت و سپس کل جدول را به یک بافر محلی انتقال داد و آن را برای استخراج اشاره‌گرهای تابع تحلیل کرد. هم‌چنین می‌توان با استفاده از پلاگین IDT در ابزار ولاتیلیتی به ورودی‌های این جدول دست یافت [۱۷].

۳-۲- استخراج اشاره‌گرهای تابع از مخزن هسته

پس از استخراج اشاره‌گرهای موجود در کد مؤلفه‌های هسته و ساختارهای شناخته‌شده سیستم‌عامل، باید اشاره‌گرهای موجود در حافظه مخزن هسته استخراج شوند. برای این امر ابتدا باید نواحی حافظه اختصاص داده شده به‌صورت پویا را استخراج کرده و سپس با جستجو بر اساس لیست اشاره‌گرهای به‌دست‌آمده از مراحل قبل، اشاره‌گرهای این ناحیه را پیدا کرد. برای دست‌یابی به این امر از دو مؤلفه جستجو و یادگیری استفاده می‌شود که در ادامه تشریح داده می‌شوند.

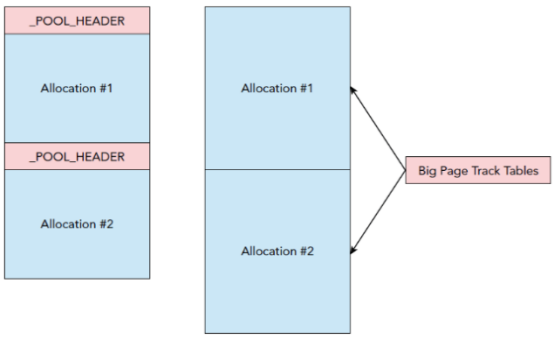
مؤلفه جستجو، حافظه مخزن هسته را استخراج کرده و اشاره‌گرهای تابع کاندید در این فضا را جستجو می‌کند. در این مرحله باید سه وظیفه کلیدی انجام شود. ابتدا باید داده‌های موجود در مخزن هسته را از فضای هسته در حافظه فیزیکی یک ماشین مجازی استخراج کرد که برای این امر ابتدا باید ساختارهای داده که به حافظه مخزن هسته ارجاع دارند و تخصیص‌های پویا در مخزن هسته را شناسایی می‌کنند، استخراج شوند. وظیفه دوم این است که اشاره‌گرهای تابع موجود در ساختارهای داده مخزن هسته را پیدا کرد که برای انجام این کار باید از لیست ساخته‌شده در مرحله اول استفاده کرد و اشاره‌گرهای تابع در مخزن هسته را که با لیست اشاره‌گرهای تابع ساخته شده در مرحله اول تطابق دارند، شناسایی کرد. وظیفه سوم این است که مشخص شود آیا اشاره‌گر کاندید شناسایی‌شده به مؤلفه یادگیری فرستاده شود و یا به‌عنوان اشاره‌گر تابع اصلی شناخته شده و به مرحله نظارت فرستاده شود. این تصمیم بستگی به ورودی‌های لیست اشاره‌گر اصلی دارد که شامل اطلاعاتی درباره اشاره‌گرهاست. این اطلاعات شامل نام مؤلفه‌ای که تخصیص دربرگیرنده اشاره‌گر تابع را ایجاد کرده است، اندازه تخصیص و آفست اشاره‌گر از آدرس پایه تخصیص است. زمانی که یک اشاره‌گر تابع در فضای مخزن پیدا شد ابتدا باید به اطلاعات آن دست یافت و سپس آن را در لیست اشاره‌گر اصلی جستجو کرد. اگر در لیست پیدا شد، اشاره‌گر به لیست اشاره‌گرهای تحت نظارت افزوده می‌شود در غیر این‌صورت اشاره‌گر به مرحله یادگیری فرستاده می‌شود.

تخصیص حافظه در فضای مخزن هسته بر اساس اندازه به دو دسته تقسیم می‌شود: حفره‌های کوچک و تخصیص‌های بزرگ، یک حفره کوچک به کمتر از یک صفحه برای تخصیص نیاز دارد و اما یک تخصیص بزرگ به بیش از یک صفحه برای یک تخصیص نیاز دارد. مایکروسافت ویندوز، این دو نوع تخصیص‌ها را در ساختارهای داده جداگانه ردیابی می‌کند. تخصیص بزرگ از طریق PoolBigPageTable ردیابی می‌شود که هر ورودی توسط یک ساختار POOL_TRACK_BIG_PAGE ارائه می‌شود. مکان PoolBigPageTable در قسمت data، از ntoskrnl.exe پیدا می‌شود و حفره‌های کوچک به‌طور کامل در یک صفحه مقیم می‌شوند و هر صفحه می‌تواند شامل چندین حفره کوچک باشد که به‌وسیله ساختارهای POOL_HEADER نمایش داده می‌شود که به‌طور مجاور در یک دنباله قرار می‌گیرند. پس ابتدا باید

[1] Absolute Address
[2] Volatility

صفحات تخصیص داده شده پیدا شوند و سپس برای استخراج حفره‌های کوچک پردازش شوند. شکل (۳) ساختار تخصیص‌های کوچک و بزرگ را در مخزن هسته نمایش می‌دهد [۱۶].

همان‌طور که قبلاً گفته شد، روت‌کیت‌ها به دنبال قلاب‌اندازی اشاره‌گرهای تابعی هستند که بتوانند تغییرات ماندگاری را در جریان کنترل سیستم ایجاد کنند و از آنجا که اشاره‌گرهای تابع موجود در مخزن صفحه‌بندی شده ممکن است به خارج از حافظه منتقل شوند بنابراین، هدف مطلوبی برای تغییر ماندگار جریان کنترل هسته نیستند.



شکل (۳): ساختار تخصیص‌های بزرگ و کوچک در حافظه سیستم‌عامل

همیشه این احتمال وجود دارد که اشاره‌گرهای تابع دستکاری‌شده به خارج از حافظه منتقل شوند که این امر حفظ کنترل جریان سیستم را مشکل‌تر می‌کند. صفحات در فضای مخزن غیرصفحه‌بندی همیشه در حافظه فیزیکی مقیم هستند بنابراین، اشاره‌گرهای موجود در این نواحی برای تغییر جریان کنترل سیستم قابل اعتمادتر هستند. از این‌رو در روش پیشنهادی، فقط اشاره‌گرهای تابع موجود در حافظه مخزن غیرصفحه‌بندی حفره‌های کوچک حافظه مخزن هسته بررسی می‌شوند. به این ترتیب هم سرعت پیمایش حافظه برای یافتن اشاره‌گرهای تابع افزایش می‌یابد و هم این‌که مهم‌ترین اشاره‌گرهای مورد هدف روت‌کیت‌ها به‌دست می‌آید.

مؤلفه یادگیری اشاره‌گرهای تابع اصلی را از داده‌های تصادفی مشخص می‌کند. یک اشاره‌گر مکان‌یابی‌شده در مرحله جستجو ممکن است اصلی باشد یا نباشد، زیرا ممکن است با یک داده تصادفی در مخزن هسته که مطابق با یک اشاره‌گر در لیست اشاره‌گرهاست اشتباه گرفته شود. زمانی‌که مؤلفه یادگیری یک اشاره‌گر تابع را دریافت می‌کند، اشاره‌گر را در طول کل محدوده حیاتش مشاهده می‌کند، مطابق [۷]، ۹۷٪ اشاره‌گرهای تابع در ویندوز در کل محدوده حیاتشان تغییر نمی‌کنند، که پایه‌ای برای شناسایی اشاره‌گرهای اصلی ارائه می‌دهد. اگر یک اشاره‌گر در لیست تا آزادسازی حافظه دربرگیرنده اشاره‌گر، تغییر نکند، مؤلفه یادگیری فرض می‌کند که آن یک اشاره‌گر تابع اصلی است. مؤلفه یادگیری، اشاره‌گرهای تابع ارسال‌شده در مرحله استخراج را دریافت می‌کند. اگر یک اشاره‌گر در طول محدوده کل حیاتش یعنی تا زمانی‌که بلاک تخصیص در برگیرنده آن آزاد می‌شود، تغییر نکند آن را به‌عنوان یک اشاره‌گر تابع اصلی در نظر می‌گیرد. به‌منظور شناسایی اعتبار فعلی یک تخصیص، آدرس پایه تخصیص می‌تواند بررسی شود اگر تخصیص بزرگ باشد. اگر محتوای آن آدرس، یک نباشد به این معنی است که یک تخصیص معتبر است. در حالت حفره‌های کوچک، مرحله یادگیری، به اولین بیت فیلد PoolType در سرآمد POOL_HEADER حفره نگاه می‌کند. اگر یک باشد به معنای معتبر بودن تخصیص است. حالت‌هایی وجود دارد که یک اشاره‌گر، محدوده حیات طولانی دارد که معیار اول را برای یادگیری کم‌موثر می‌کند. بنابراین، مرحله یادگیری، یک مقدار آستانه بر اساس سن اشاره‌گر استفاده می‌کند. اگر سن از مقدار آستانه تجاوز کند، به‌عنوان اشاره‌گر اصلی معرفی می‌شود. روش پیشنهادی، یکپارچگی اشاره‌گرهای تابع که در محدوده حیاتشان تغییر می‌کنند را بررسی نمی‌کند. منطق این است که هدف مطلوب برای حمله، اشاره‌گرهایی هستند که اولاً محدوده حیات طولانی دارند و ثانیاً در طول محدوده حیاتشان تغییر نمی‌کنند. این امر به یک حمله‌کننده اجازه می‌دهد که یک تغییر ماندگار در جریان کنترل سیستم ایجاد کند. چنین اشاره‌گرهایی ممکن است برای مدت طولانی در سیستم زندگی کنند که ظاهراً فرآیند یادگیری را آهسته می‌کنند به‌طوری‌که نمی‌توان تصمیم گرفت آیا اشاره‌گر اصلی است تا زمانی‌که تخصیص در برگیرنده آن آزاد نشود. بنابراین، به‌منظور شناسایی چنین اشاره‌گرهایی به‌طور مؤثر، مؤلفه یادگیری، سن اشاره‌گر کاندید را نظارت می‌کند اگر سن از آستانه تجاوز کند، به‌عنوان اصلی در نظر گرفته می‌شود. لیست اشاره‌گرهای یادگیری شامل زمان ایجاد یک ورودی اشاره‌گر در لیست است که مؤلفه یادگیری از آن برای پیش‌بینی سن اشاره‌گر استفاده می‌کند.

۳-۳- نظارت یکپارچگی اشاره‌گرهای تابع

پس از استخراج اشاره‌گرهای تابع از منابع سیستم‌عامل و فضای آدرس مخزن هسته، این اشاره‌گرهای اصلی استخراج‌شده به همراه آدرس دربرگیرنده آن‌ها به مرحله نظارت فرستاده می‌شوند. هدف از نظارت، در واقع بررسی یکپارچگی اشاره‌گرهای تابع هسته و ایجاد هشدار در زمان تغییر آن‌ها توسط روت‌کیت‌ها است. در مرحله نظارت، یک لیست از اشاره‌گرهای تابعی که باید نظارت شوند نگهداری می‌شود که شامل مقدار این اشاره‌گرها و آدرس دربرگیرنده آن‌ها است. معمولاً اشاره‌گرهای تابع مورد نظارت تغییر نمی‌کنند، هرچند که اگر مقدار اشاره‌گر به مقدار دیگری در محدوده آدرس کد هسته یا به مقدار null تغییر کند یا از null به مقدار دیگر در محدوده آدرس کد هسته تغییر کند،

این تغییرات مجاز در نظر گرفته می‌شوند. در بقیه حالت‌ها تغییر مجاز نبوده و نشان‌دهنده دست‌کاری غیرمجاز توسط روت‌کیت‌ها است. اشاره‌گرهای تابع موجود در لیست اشاره‌گرهای مورد نظارت تا زمانی‌که تخصیص دربرگیرنده آن‌ها آزاد می‌شود، نظارت می‌شوند. زمانی‌که تخصیص دربرگیرنده اشاره‌گر تابعی آزاد می‌شود، نظارت اشاره‌گرهای تابع موجود در آن، متوقف شده و اشاره‌گر از لیست اشاره‌گرهای مورد نظارت حذف می‌شود.

۴- ارزیابی روش پیشنهادی

در اینجا، سه جنبه از روش ارائه‌شده مورد ارزیابی قرار می‌گیرد، در ابتدا یک مجموعه روت‌کیت شناخته‌شده سطح هسته اجرا می‌شوند تا مؤثر بودن روش در شناسایی روت‌کیت‌ها نشان داده شود، سپس زمان اجرای روش با سایر ابزارهای کشف روت‌کیت مقایسه می‌شود و در نهایت سربار کارایی روش اندازه گرفته می‌شود. در ادامه کار، یک سری نمونه روت‌کیت شناخته شده سطح هسته از منابع عمومی [۱۴] و تحقیقات انجام‌شده در این زمینه برای ارزیابی انتخاب شده‌اند. نمونه‌های روت‌کیتی که برای ارزیابی انتخاب شده‌اند در زمینه نصب قلاب‌اندازی‌های سطح هسته شناخته شده هستند و قادر به اجرا در محیط آزمایش هستند. برای مقایسه با روش ارائه‌شده، ابزارهای Hookscout [۲]، IceSword [8]، VICE [۷ و ۱۲] HookFinder انتخاب شده‌اند که در قسمت کارهای مرتبط، معرفی شدند.

۴-۱- کشف قلاب‌اندازی

در جدول (۱) روش ارائه‌شده با ابزارهای ذکرشده در قسمت قبل مقایسه شده‌اند. برای ابزار Hookscout از حرف اختصاری H، ابزار IceSword حرف اختصاری I، ابزار VICE حرف اختصاری V، ابزار HookFinder حرف اختصاری F و روش پیشنهادی حرف اختصاری P به‌کار رفته است.

جدول (۱): مقایسه روش پیشنهادی با سایر ابزارها در کشف قلاب‌اندازی

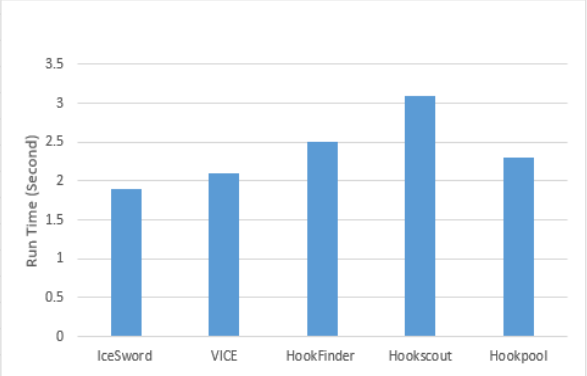
نمونه روت‌کیت	محل قلاب‌اندازی	I	V	F	H	P
HideProcessHookMDL	SSDT	*	*	*	*	*
Sony Rootkit	SSDT	*	*	*	*	*
Storm Worm	SSDT	*	*	*	*	*
Shadow Walker	IDT		*	*	*	*
basic interrupt 3	IDT		*	*	*	*
TCPIRPHOOK	Tcp driver object		*	*	*	*
Rustock.C	Fastfat driver object			*	*	*
Uay Backdoor	NDIS data block			*	*	*
KBD_hook	pool data region				*	*
Agobot	Ntoskrnl EAT			*		*

نتایج به‌دست‌آمده از اجرای این ابزارها در جدول نمایش داده شده است. همان‌طور که مشخص است همه ابزارها، قلاب‌اندازی SSDT را شناسایی می‌کنند، قلاب‌اندازی جدول IDT را هم جز ابزار IceSword، دیگر ابزارها شناسایی می‌کنند. روت‌کیت‌های TCPIRPHOOK و Rustock.C، اشاره‌گرهای تابع موجود در اشیاء دیوایس درایور Tcp و Fastfat را قلاب‌اندازی می‌کنند. IceSword اشیاء هسته را مورد بررسی قرار نمی‌دهد بنابراین، نمی‌تواند قلاب‌اندازی‌های انجام‌شده در اشیاء هسته را شناسایی کند. ابزار HookFinder هردو شیء را بررسی می‌کند و ابزار VICE تنها شیء Fastfat را بررسی می‌کند. Uay Backdoor اشاره‌گرهای تابع موجود در ساختار داده NDIS نگهداری شده برای پروتکل TCP/IP را قلاب‌اندازی می‌کند. IceSword و VICE قادر به شناسایی این قلاب‌اندازی نیستند اما ابزار HookFinder به علت بررسی لیست پروتکل‌های شبکه قادر به شناسایی این نوع قلاب‌اندازی است. روت‌کیت KBD_hook اشاره‌گرهای تابع موجود در حافظه پویای مربوط به درایور صفحه کلید را قلاب‌اندازی می‌کند که فقط ابزار hookscout و روش ارائه‌شده قادر به شناسایی آن هستند. روت‌کیت Agobot همان‌طور که قبلاً ذکر شد می‌تواند بسیاری از ابزارهای امنیتی ماشین هدف را شناسایی کند و در نتیجه این ابزارها قادر به شناسایی آن نیستند، اما از آنجا که ابزار ارائه‌شده در این پژوهش و هم‌چنین HookFinder در سطح ناظر ممتاز، سیستم هدف را درون‌بینی و نظارت می‌کنند نمی‌توانند توسط این روت‌کیت شناسایی شده و در نتیجه روت‌کیت توسط این ابزارها شناسایی می‌شود. در مقایسه با ابزارهای ارائه‌شده، روش پیشنهادی قادر به شناسایی همه نمونه‌های روت‌کیت است. تفاوت کلیدی روش پیشنهادی با روش‌های دیگر در این است که اشاره‌گرهای تابع مهمی که بیشترین هدف روت‌کیت‌ها هستند را هم در ساختارهای داده ثابت هسته و هم در حافظه پویای هسته، مورد بررسی قرار می‌دهد. از طرف دیگر این بررسی‌ها در سطح ناظر ممتاز با استفاده از روش درون‌بینی ماشین مجازی انجام می‌شود که این امر سبب می‌شود بدافزارها نتوانند آن را شناسایی کنند و مانع فعالیتش شوند.

۴-۲- زمان اجرا

در ادامه آزمایش‌ها، زمان اجرای روش پیشنهادی، با ابزارهای معرفی‌شده در قسمت قبل مقایسه می‌شوند. روش ارائه‌شده با نام Hookpool در نمودار شکل (۴) نمایش داده شده است. همان‌طور که مشخص است ابزارهای IceSword, VICE زمان اجرای کمتری نیاز دارند و دلیل این امر این است که این ابزارها، کل حافظه هسته را برای پیدا کردن اشیاء هسته پویش نمی‌کنند و

بنابراین، زمان کمتری برای پویش حافظه صرف می‌کنند. روش پیشنهادی به‌دلیل این‌که فضای جستجوی مخزن هسته را کاهش داده و فقط در صفحات حافظه غیرصفحه‌بندی جستجو انجام می‌دهد، از نظر زمان اجرا از دو روش دیگر بهتر است.



شکل (۴): نمودار مقایسه زمان اجرای روش پیشنهادی با دیگر ابزارها

۴-۳- سربار کارایی

برای مشاهده چگونگی تأثیر روش پیشنهادی بر کارایی سیستم، چندین حجم کاری اجرا می‌شوند و زمان‌های اجرایی آن‌ها در حالت اجرای روش ارائه‌شده و بدون آن، اندازه گرفته می‌شوند. هم‌چنین کارایی با دو فاصله زمانی متفاوت (۱ ثانیه و ۵ ثانیه) اندازه گرفته می‌شود. حجم کاری شامل کپی کردن یک ساختار دایرکتوری، اجرای فشرده‌سازی با 7zip و دانلود یک فایل با wget است. اندازه کل ساختار دایرکتوری ۴۵ مگابایت و اندازه فایل دانلودشده ۳۰ مگابایت است. جدول (۲) زمان اجرا برای هر حجم کاری را نمایش می‌دهد. هر حجم کاری ۷ بار اجرا شده و میانگین آن محاسبه می‌شود. نتایج نشان می‌دهد کاهش ایجاد شده توسط روش ارائه شده، برای فواصل بررسی ۵ ثانیه و ۱ ثانیه به ترتیب ۴/۶ درصد و ۵ درصد است.

جدول (۲): سربار عملکرد روش ارائه‌شده در چندین حجم کاری

بار کاری	بدون اجرای ابزار پیشنهادی	اجرای ابزار پیشنهادی		درصد کاهش	
		ثانیه اول اجرا	ثانیه پنجم اجرا	ثانیه اول اجرا	ثانیه پنجم اجرا
کپی کردن دایرکتوری	۱۸/۳۲	۱۹/۲۴	۱۹/۱۹	٪۴/۸	٪۴/۶
فشرده‌سازی فایل	۲۱	۲۲/۱۰	۲۲	٪۵	٪۴/۶
دانلود فایل	۴۵/۱۸	۴۷/۵۲	۴۷/۲۳	٪۵	٪۴/۴

۵- نتیجه‌گیری

در این پژوهش، کارهای انجام شده برای مقابله با روت‌کیت‌های سطح هسته که با تغییر جریان کنترل سیستم، سعی در پنهان‌سازی خود و بدافزارهایشان دارند، بررسی شدند و مشخص شد که این روش‌ها دارای نقاط ضعفی از قبیل وابستگی به کد منبع هسته سیستم‌عامل، اجرا در سطح امتیاز یکسان با بدافزار، سربار عملکردی قابل توجه و عدم بررسی فضای پویای هسته در شناسایی روت‌کیت‌های جدید هستند. برای غلبه بر مشکلات مطرح‌شده، روشی ارائه شد که در سطح ناظر ممتاز اجرا شده و با استخراج اشاره‌گرهای تابع موجود در منابع سیستم‌عامل و حافظه مخزن هسته، یکپارچگی آن‌ها را نظارت می‌کند. روش پیشنهادی با استخراج و نظارت اشاره‌گرهای تابع در نواحی مهم حافظه هسته سیستم که بیشتر هدف روت‌کیت‌ها هستند، قادر به شناسایی قلاب‌اندازی نصب‌شده در این نواحی توسط روت‌کیت‌های پیشرفته است. هم‌چنین روش ارائه‌شده در سطح ناظر ممتاز اجرا شده و این امر ردیابی و شناسایی آن توسط روت‌کیت‌ها را مشکل می‌کند و روش ارائه‌شده می‌تواند بدون دخالت روت‌کیت‌ها، ماشین مجازی در حال اجرا را نظارت کند. ارزیابی‌های انجام‌شده نشان می‌دهد روش ارائه‌شده با کارایی قابل قبولی قادر به کشف همه نمونه روت‌کیت‌ها است.

۶- منابع

[1] C.-M. Chen, et al., "A Methodology for Hook-Based Kernel Level Rootkits," International Conference on Information Security Practice and Experience, Springer International Publishing, 2014.

[2] Z. Wang, et al., "|Countering persistent kernel rootkits through systematic hook discovery," International Workshop on Recent Advances in Intrusion Detection, Springer Berlin Heidelberg, 2008.

[3] H. Yin, et al., "HookScout: proactive binary-centric hook detection," International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, Springer Berlin Heidelberg, 2010.

[4] G. Yan, et al., "MOSKG: countering kernel rootkits with a secure paging mechanism," Security and Communication Networks 8.18, pp. 3580-3591, 2015.

[5] S. Vomel and L. Hermann, "Visualizing indicators of Rootkit infections in memory forensics," IT Security Incident Management and IT Forensics (IMF), 2013 Seventh International Conference on. IEEE, 2013.

[5] M. Carbone, et al., "Mapping kernel objects to enable systematic integrity checking," Proceedings of the 16th ACM conference on Computer and communications security, ACM, 2009.

[6] J. Butler and H. Greg, "VICE-Catch the hookers!(Plus new rootkit techniques)," Black Hat USA 2004 Conference, Las Vegas, USA. 2004.

[7] IceSword, http://www.antirootkit.com/software/IceSword.htm

[8] Jr. Petroni, L. Nick, and M. Hicks, "Automated detection of persistent kernel control-flow attacks," Proceedings of the 14th ACM conference on Computer and communications security, ACM, 2007.

[9] A. Baliga, V. Ganapathy, and L. Iftode, "Automatic Inference and Enforcement of Kernel Data tructure Invariants," In Pro-ceedings of the 24th Annual Computer Security Applications Conference (ACSAC 2008), Anaheim, California, USA, pp. 77-86, 2008.

[10] http://www.sans.org/course/memory-forensics-in-depth.%22memory-forensics-in-depth%20%22.2014

[11] Z. Wang, X. Jiang, W. Cui, and P. Ning, "Countering Kernel Rootkits with Lightweight Hook Protection," In Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS 2009), Chicago, IL, USA, pp. 545-554, 2009.

[12] F. Yangchun, Z. Lin, and D. Brumley, "Automatically deriving pointer reference expressions from binary code for memory dump analysis," Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ACM, 2015.

[13] C. Weng, et al., "CloudMon: Monitoring Virtual Machines in Clouds," IEEE Transactions on Computers 65.12, pp. 3787-3793, 2016.

[14] A. Bianchi, et al., "Blacksheep: detecting compromised hosts in homogeneous crowds," Proceedings of the 2012 ACM conference on Computer and communications security, ACM, 2012.

[15] H. Yin, Z. Liang, and D. Song, "|HookFinder: Identifying and understanding malware hooking behaviors," In Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS'08), February 2008.

[16] I. Ahmed, et al., "Integrity checking of function pointers in kernel pools via virtual machine introspection," Information Security, Springer International Publishing, pp. 3-19, 2015.

[17] S. Sparks, E. Shawn, and Z. Cliff, "Windows Rootkits-a Game of Hide and Seek," Handbook of Security and Networks, vol. 345, 2011.

[18] Y. Liu, et al., "Concurrent and consistent virtual machine introspection with hardware transactional memory," 2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA), IEEE, 2014.

[19] A. Prakash, et al., "On the Trustworthiness of Memory Analysis—An Empirical Study from the Perspective of Binary Execution," IEEE Transactions on Dependable and Secure Computing 12.5, pp. 557-570, 2015.

[20] M. H. Ligh, A. Case, J. Levy, and A. Walters, "The Art of Memory Forensics: Detecting Malware and Threats in Windows, Linux, and Mac Memory," John Wiley and Sons, 2014.

[21] rootkit.com, http://www.rootkit.com/

An Approach to Rootkit Detection Based on Virtual Machine Introspection

S. Parsa[*], F. Jamshidinia

Abstract

Kernel rootkits have posed serious security threats due to their stealthy manner. To hide their presence and activities, many rootkits hijack control flows by modifying control data or hooks in the kernel space function pointers, especially those dynamically allocated from heaps and memory pools. These areas of kernel memory are currently not monitored by kernel integrity checkers. On the other hand, Traditional host-based detection tools execute inside the host they are protecting, therefore, since these tools execute within the kernel, they could be easily detected by the rootkits. To solve this problem, Current rootkit detection tools deploy virtual machine introspection technique that monitors the state of running virtual machine at hypervisor level, without rootkits interposition. The goal of this thesis is to present an approach based on virtual machine introspection, to detect rootkits which hide themselves and their associated malwares in main memory using modifying system control flow. The proposed approach monitors the integrity of Windows kernel function pointers that are potentially prone to malicious exploits, based entirely on virtual machine introspection. The approache is evaluated with a set of rootkits which use advanced hooking techniques and show that it detects all of the stealth techniques utilized

Key Words: *Rootkit, Virtual Machine Introspection, Hooking, Function Pointer*

[*] Iran University of Science & Technology - (parsa@iust.ac.ir) - Writer-in-Charge