هفتمین کنفرانس بین المللی انجمن رمز ایران

۲۴ و ۲۵ شهریورماه ۱۳۸۹، دانشگاه صنعتی خواجه نصیرالدین طوسی

انجمن رمزایران

دانشگاه صنعتی خواجه نصیرالدین طوسی

تشخیص نفوذ با استفاده از رویدادنگاری فراخوان های سیستمی در محیط مانیتور ماشین مجازی

حامد نعمتی[۱]، رضا عزمی[۲]، علیرضا قهرمانیان[۳] و محمدتقی میرمحمدرضایی[۴]

دانشگاه صنعتی مالک اشتر[۱]

hnsnemati@gmail.com

دانشگاه الزهراء [۲]

azmi@alzahra.ac.i

دانشگاه صنعتی مالک اشتر[۳]

mat2ag@yahoo.com

دانشگاه الزهراء [۴]

mmrezaie@comp.iust.ac.ir

چکیده

رویدادنگاری فراخوان‌های سیستمی به عنوان ابزاری متداول برای پیاده‌سازی مکانیزم‌های امنیتی شناخته می‌شود. طی چند دهه اخیر راهکارهای مختلفی برای تشخیص نفوذ براساس رویدادنگاری فراخوان‌های سیستمی ارائه شده ولی همزمان با پیشرفت این مکانیزم‌ها، نفوذگران تلاش نمودند، شیوه رویدادنگاری فراخوان‌های سیستمی را تغییر دهند تا بتوانند حضور خود در سیستم و نوع فعالیتی که انجام می دهند را مخفی نمایند. در این مقاله سعی بر آن است تا با معرفی یک معماری جدید مبتنی بر مانیتور ماشین مجازی (ناظر سیستم)، مکانیزمی جهت تضمین سلامت رویدادنگاری فراخوان‌های سیستمی ارائه شود. در ادامه و با توجه به حجم فراوان داده‌های رویدادنگاری شده، طرحی براساس درخت هافمن برای تحلیل و فشرده سازی فایل رویدادنگاری ارائه می شود. در مرحله تشخیص نفوذ، با استفاده از شبکه بیزین ناهنجاری‌های داده‌های گردآوری شده، مشخص می‌شود. برای ارزیابی معماری ارائه شده، نمونه اولیه ای مبتنی بر مانیتور ماشین مجازی SCInterceptor پیاده‌سازی شده است.

واژه های کلیدی

شخیص نفوذ، مانیتور ماشین مجازی، رویدادنگاری، فراخوان‌های سیستمی، درخت هافمن، دسته‌بندی کننده بیزین.

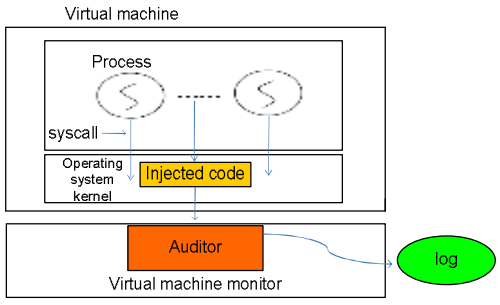
۱- مقدمه

در اکثر سیستم‌های نرم‌افزاری بزرگ و پیچیده احتمال آسیب‌پذیری وجود دارد و با استفاده از تمام امکانات نرم‌افزاری امروزی از حذف کامل راه‌های نفوذ به این سیستم‌ها نمی‌توان اطمینان حاصل کرد. سیستم‌های‌عامل از جمله نرم‌افزارهای بزرگ و پیچیده و به طبع آن آسیب‌پذیر هستند. از جمله روش‌های تشخیص نفوذ در سیستم‌های‌عامل، رویدادنگاری فراخوان‌های سیستمی می‌باشد که به صورت کلی در سطح هسته انجام می‌شود. در این مقاله روشی نوین برای رویدادنگاری فراخوانی‌های سیستمی در لایه مانیتور ماشین مجازی ارائه می‌دهیم که نسبت به راهکارهای دیگران به دلیل استفاده از تکنولوژی‌های مجازی‌سازی پردازنده‌ها در برابر حملات مقاوم‌تر بوده و در عین حال کارایی مناسبی دارد.

فراخوان سیستم مکانیزمی است که یک برنامه کاربردی توسط آن سرویسی را از سیستم‌عامل در خواست می‌کند. تنها دروازه دسترسی به منابع سیستم و سخت افزارها فراخوان های سیستمی هستند. فراخوانی این توابع موجب می‌شود که ابتدا درخواست، اعتبار سنجی شده سپس در اختیار گرداننده مناسب قرار گیرد. فراخوان‌های سیستمی برای اجرا از دستورات خاصی از سیستم‌عامل استفاده می‌کنند که منجر به انتقال حالت اجرا بین دو مد هسته و کاربر می‌گردد. پس از فراخوانی یک فراخوان سیستمی اجرای برنامه تا پایان اجرای این فراخوان سیستمی دچار وقفه گردیده و پس از پایان آن دوباره کنترل به برنامه باز می‌گردد. پیاده‌سازی فراخوان‌های سیستمی با توجه به معماری‌های مختلف، می‌تواند متفاوت باشد. یک روش پیاده‌سازی فراخوان‌های سیستمی استفاده از وقفه‌های نرم‌افزاری است که در سیستم عامل لینوکس شماره فراخوان سیستمی مربوطه در ثبات EAX [1] قرار گرفته سپس وقفه 0x80 فرا خوانی می‌گردد.

رویدادنگاری فراخوان‌های سیستمی که توسط یک پردازه فراخوانی می شوند الگوی مناسبی از چگونگی رفتار آن پردازه فراهم می‌آورد. به همین دلیل پروژه‌های مختلفی در لایه هسته از رویدادنگاری فراخوانی‌های سیستمی استفاده کرده‌اند[14][15]. این روش‌ها از فراخوان‌های سیستمی به عنوان امضایی از چگونگی رفتار پردازه استفاده می‌کنند تا رفتار نرمال یا غیر نرمال سیستم را تشخیص دهند. در لایه ناظر سیستم نیز از فراخوانی‌های سیستمی به عنوان امضایی از چگونگی رفتار پردازه‌ها استفاده شده است[2][3][6].

با توجه به حجم فراوان داده‌های رویدادنگاری شده و وجود موارد افزونه زیاد در یک فایل رویدادنگاری، در این مقاله با استفاده از ساختار درخت هافمن تلاش کرده‌ایم تا با حفظ جامعیت داده‌های جمع‌آوری شده، ساختار مناسبی را برای فایل رویدادنگاری فراهم آوریم. در مرحله بعد برای دسته بندی داده های نرمال و غیر نرمال و جداسازی آنها از شبکه‌های بیزین مبتنی بر تئوری اطلاعات [10]، [12] استفاده می‌کنیم.



شکل 1. معماری کلی سیستم پیشنهادی

برای ارزیابی معماری پیشنهادی (شکل1و2) مانیتور ماشین مجازی SCInterceptor برپایه پروژه Bitvisor[4] پیاده‌سازی شده است. ناظر سیستم Bitvisor مانیتور ماشین مجازی‌ای است که با رویکرد امنیتی طراحی شده است. معماری کلی این مانیتور با عنوان وساطت جزئی[2] شناخته می‌شود که به وساطت و تحلیل امنیتی بخش خاصی از تعاملات (برای مثال درخواست‌های ورودی/خروجی) بین سیستم‌عامل مهمان با سخت افزار می‌پردازد و مابقی درخواست‌ها مستقیما به سخت افزار منتقل می‌شوند. به خاطر معماری ویژه Bitvisor امکان اجرای یک ماشین مجازی بر روی این مانیتور در هر زمان وجود دارد.

در ادامه و در بخش 2 مروری بر کارهای گذشته خواهیم داشت. در بخش3 معماری پیشنهادی و شیوه پیاده‌سازی آن را توصیف خواهیم نمود. در بخش 4 ارزیابی معماری پیشنهادی و در بخش 5 نتایج مبحث ارائه شده است.

2- مروری بر کارهای دیگران

رویدادنگاری فراخوانی‌های سیستم را می‌توان در لایه‌های مختلف سیستمی (ناظر و هسته) انجام داد. در ادامه به بررسی پروژه‌های مختلفی که در این لایه‌ها پیاده‌سازی شده‌اند می‌پردازیم.

در لایه هسته می‌توان به تمامی زیرساخت‌های هسته دسترسی داشت و عدم وجود تغییر مد به دلیل وجود همه زیرساخت‌ها در یک مد ، سربار را کاهش می‌دهد. [14]، [15] و [16] مثال‌هایی از پروژه‌های انجام شده در این لایه هستند. همه این پروژه‌ها از وساطت برای نظارت بر فراخوانی‌ها استفاده می‌کنند.

در لایه ناظر سیستم به دلیل استفاده از تکنولوژی سخت‌افزاری مجازی‌سازی، سربار تغییر مد بین ناظر و سیستم‌عامل کمتر از تغییر مد بین لایه سیستم‌عامل و کاربر می‌باشد[19]. ناظر IntroVirt[6] که بر پایه پروژه UML[20] پیاده‌سازی شده سیستمی است که نفوذها را از خارج ماشین مجازی تشخیص داده و پاسخ مناسبی را در برابر آن ارائه می‌کند.

در [3]، Xuxian Jiang و Xinyuan Wang سعی نمودند با ارائه یک معماری مبتنی بر مانیتور ماشین مجازی QEMU[18]، Honeypotای براساس رویدادنگاری فراخوان‌های سیستمی پیاده‌سازی نمایند. آن‌ها نمونه اولیه از معماری پیشنهادی خود را با نام VMScope پیاده‌سازی نمودند.

[1] Register

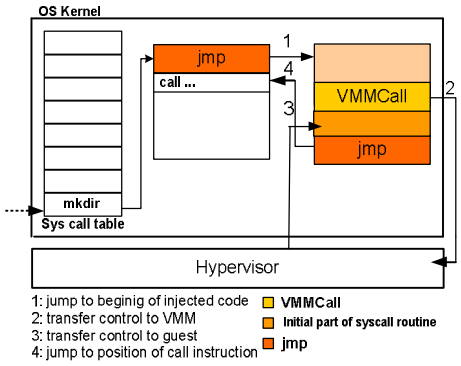
[2] Parapass-through

در سال 2008، Koichi Onoue در[2] با ارائه یک معماری مبتنی بر مانیتور ماشین مجازی Xen به صورت مجازی‌سازی جزئی[1] سعی نمود تا پردازه‌های در حال اجرا بر روی سیستم‌عامل مهمان را کنترل نماید.

هر کدام از کارهایی که در این بخش عنوان شدند نماینده یک نوع خاص از معماری برای پیاده‌سازی رویدادنگاری می‌باشند. در ادامه به توصیف معماری پیشنهادی خود و شیوه پیاده‌سازی آن خواهیم پرداخت.

3- معماری و پیاده‌سازی

معماری کلی SCInterceptor به این صورت است در زمان فراخوانی فراخوان‌های سیستمی کنترل به ناظر داده می‌شود. ناظر رویدادنگاری می‌کند و کنترل را دوباره به سیستم‌عامل باز می‌گرداند.



شکل 2. ساختار درونی معماری SCInterceptor

پیاده‌سازی فراخوانی سیستمی در سیستم‌های‌عامل در معماری اینتل از طریق دو مکانیزم، وقفه نرم‌افزاری 0x80 و SysEnter/SysExit برای معماری X86 (SysCall/SysRet برای معماری AMD64) انجام می‌شود. باید توجه نمود که وجود مکانیزم‌های مختلف برای معماری‌های متفاوت سخت‌افزاری عمل رویدادنگاری فراخوانی سیستمی را مشکل می‌سازد.

3-1- جمع‌آوری اطلاعات

با هدف اینکه رویدادنگاری بدون تغییر کد سیستم عامل انجام شود و البته رویکردهای مختلف ارائه شده برای پیاده‌سازی فراخوانی‌های سیستم‌عامل پشتیبانی شوند؛ عملیات رویدادنگاری توسط SCInterceptor در 3 فاز انجام می‌شود.

فاز 1: شناسایی

در اولین قدم باید IDT Base و IDT Limit محاسبه شوند. برای این منظور از ساختار VMCS[2] استفاده می‌شود. با استفاده از آدرس بدست آمده می‌توان مقادیر High/Mid/Low Offset را بدست آورد. با ترکیب این مقادیر مقدارSyscall Offset بدست می‌آید که آدرس شروع تابع System_call() را می‌توان از افزودن آن به مقدار Base مدخل 0x60 جدول GDT بدست آورد.

برای بدست آوردن آدرس جدول توزیع فراخوانی‌های سیستم بر اساس الگوی "\xff\x14\x85" از آدرس بدست آمده در مرحله قبل شروع به جستجو می‌کنیم. 4 بایت بعدی مقدار بدست آمده، آدرس جدول مورد نظر می‌باشد.

برای ایجاد فضایی برای تزریق کد در هسته سیستم‌عامل، نیاز به آدرس تابع kmalloc() داریم. برای یافتن این آدرس با جستجوی الگوی "0__kmalloc0" در قطعه کد هسته آدرس ارجاع تابع و با جستجوی این آدرس، آدرس ابتدای تابع بدست می‌آید.

در مرحله بعد می‌بایست با فراخوانی تابع kmalloc برای کد تزریقی حافظه‌ای تخصیص داده و آدرس آن را به ناظر به منظور تزریق کد اطلاع داد. این عمل با استفاده از جانویسی روال سرویس یکی از فراخوان‌های سیستم قابل انجام است و بعد از آن کد قبلی روال دوباره جاگذاری می‌شود.

فاز 2: تزریق

در این مرحله ابتدا و انتهای یک فراخوانی سیستم را بدست می‌آوریم و آدرس فراخوانی اصلی روال(اولین دستور call) را پیدا می‌کنیم. از آدرس شروع تا آدرس مذکور را با تکه کدی که کد تزریق شده ما را فراخوانی می‌کند، جایگزین می‌کنیم ولی کد بعد از آن آدرس را دست نخورده باقی می‌گذاریم.

ساختار کد تزریق شده

ساختار کد تزریق شده (شکل 2) به ترتیب به صورت زیر می‌باشد:

1. ذخیره مقادیر ثبات‌ها در بخشی از حافظه تخصیص داده شده
2. در گام بعد فراخوانی های سیستمی getpid , getuid, getgid, geteuid, getegid برای بدست آوردن pid, uid, effective uid , effective gid, gid فراخوانی خواهند شد، مقادیر بازگشتی از طریق ثبات EAX در بخش دیگری از فضای تخصیص داده شده ذخیره خواهد

[1] Para-virtualization

[2] Virtual Machine Control Structure

شد و سپس فراخوانی ناظر با استفاده از VMM_CALL برای رویدادنگاری پارامترها مورد نظر صورت می‌گیرد.

3. اجرای بخش آغازین کد روالی که ابتدای آن جانویسی شده.
4. بازگشت به ابتدای فراخوان اصلی روال فراخوان سیستم

فاز 3: پایان

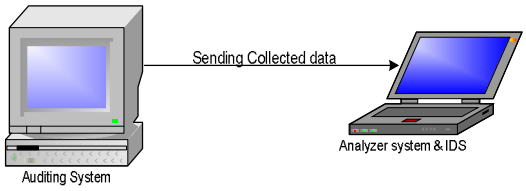
در این فاز باید وضعیت مقادیر پارامترهای سیستم را به حالت قبلی سیستم در حال اجرا باز گردانیم. فعالیت‌هایی که در این فاز انجام می‌شوند عبارتند از:

بازخوانی مقادیر ذخیره شده از حافظه

ذخیره نوع فراخوان سیستمی و کلیه پارامترهای آن

بار کردن دوباره این مقادیر در ثبات‌ها

ارسال داده ها به کامپیوتر تحلیل گر و برگشت به ادامه کار.



شکل 3: طرح کلی سیستم

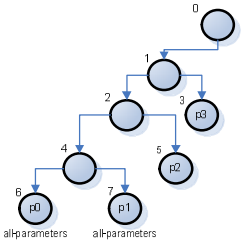
برای ارسال اطلاعات به کامپیوتر مقصد می‌توان یکی از دو مسیر درگاه سریال یا شبکه را به کار برد. به دلیل آنکه همه درگاه‌های ورودی و خروجی در اختیار ناظر است، مانیتور ماشین مجازی می‌تواند فقط تعداد مشخصی از درگاه‌ها را برای سیستم‌عامل مهمان قابل مشاهده نموده و مابقی را از دسترس آن خارج نماید. ما در SCInterceptor از این ویژگی استفاده کرده‌ایم و درگاه‌هایی که برای رویدادنگاری استفاده می‌شوند را از دید سیستم‌عامل کاملا مخفی می‌کنیم. این کار امنیت مکانیزم ارسال داده به کامپیوتر مقصد را تا حدود زیادی تامین می‌کند(شکل 3).

3-2- تحلیل فایل رویدادنگاری

از آنجا که حجم عظیم داده‌های جمع‌آوری شده در یک بازه زمانی و وجود موارد افزونه فراوان در آن‌ها، باعث بروز مشکلاتی از جمله افزایش زمان اجرایی در مرحله تشخیص نفوذ می‌شود؛ برای رفع این مشکل نیازمند روشی برای بازآرایی فایل رویدادنگاری هستیم؛ که هیچ یک از داده‌های جمع‌آوری شده را حذف یا تخریب نکند و هم اینکه دارای ساختار مشخصی باشد تا در شبکه‌های بیزینی (مرحله تشخیص‌نفوذ) قابل استفاده باشد. به همین دلایل از ساختار درخت هافمن[17] در مرحله تحلیل فایل استفاده می‌کنیم.

پیش از ایجاد درخت هافمن باید داده‌های جمع‌آوری شده را خلاصه سازی کنیم، چرا که ما از همه پارامترهای فراخوان‌های سیستمی در معماری پیشنهادی خود استفاده نمی‌کنیم، بلکه با بررسی‌های انجام شده به این نتیجه رسیده‌ایم که فقط از پارامترهای طول بافر ، مد دسترسی و پرچم ها (buffer-size, mode, flags) استفاده نماییم.

برای ایجاد درخت هافمن ابتدا احتمال وقوع هر بسته اطلاعاتی را با استفاده از فرکانس آن بسته و تعداد کل بسته‌های موجود محاسبه می‌کنیم. به عبارت دیگر ما درخت هافمن را با استفاده از احتمال وقوع هر عنصر ایجاد می‌نمائیم. دلیل استفاده از احتمال برای ایجاد درخت این است که احتمال هر عنصر خود موجودیتی نرمال‌سازی‌شده می‌باشد و برای استفاده در یادگیری شبکه بیزین مناسب‌تر است.



شکل 4: طرح کلی درخت هافمن بکار رفته

در گام دوم برای اینکه به هر عنصر اطلاعاتی در درخت برچسب منحصر بفردی اختصاص دهیم درخت را از بالا به پایین و از چپ به راست با شروع از عدد صفر برای ریشه تا انتها شماره‌گذاری کرده و سپس جهت نرمال‌سازی این اعداد تمامی آن‌ها را بر بزرگترین عدد اختصاص داده شده تقسیم می‌کنیم. بعد از انجام این عمل درخت هافمن براساس بسته‌های داده‌ای ایجاد شده، ساخته می‌شود. طرح کلی این درخت در شکل 4 قابل مشاهده است.

3-3- مرحله تشخیص نفوذ

فاز تشخیص نفوذ به صورت آفلاین[1] مبتنی بر تحلیل فایل رویدادنگاری انجام می‌شود. سامانه تشخیص در این معماری یک شبکه بیزین مبتنی بر تئوری اطلاعات است.

شبکه بیزین، روشی سیستماتیک برای تصمیم‌گیری و دسته‌بندی عناصر در شرایط عدم قطعیت می‌باشد و باید در فاز یادگیری با نمونه‌هایی از کلاس‌های دسته‌بندی عناصر آموزش داده شود.

برای آزمون و یادگیری دسته‌بندی‌کننده شبکه بیزین از تعدادی روتکیت[2] و کرم استفاده کرده و فراخوان‌های سیستمی مورد نظر را رویدادنگاری می‌کنیم. در این میان کاربران مختلف

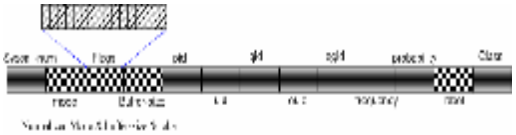
[1] Off-line

[2] rootkit

با سطوح دسترسی متفاوت اعمال متداول خود را نیز در سیستم به منظور بدست آوردن رفتار معمول سیستم، انجام می‌دادند.

به طور معمول بدافزارها در ساختار خود دارای بخش‌هایی هستند که تلاش می‌کنند آن‌ها را از دید کاربران و دیگر سامانه‌های تشخیص نفوذ مخفی کند. از این رو باید راهکاری بیابیم تا بتوانیم پردازه‌های مخرب را از پردازه‌های نرمال سیستمی تشخیص دهیم. برای رسیدن به این مقصود برنامه chkrootkitبسیار کمک کننده می‌باشد.

ابزار chkrootkit در سطح کاربر امضای بدافزارهای شناخته شده را جستجو می‌کند. این ابزار PID پردازه‌های مخرب و مخفی را بدست می‌آورد. با کسب اطمینان از این که شناسه‌ها متعلق به برنامه‌های عادی سیستم نباشند، می‌توان PIDهای بدست آمده را برای کلاس‌بندی رکوردهای رویدادنگاری‌شده استفاده کرد.



شکل 5: بسته مورد استفاده در شبکه بیزین

در مرحله بعد رکوردهای پردازه‌هایی که مخرب اعلام شده‌اند را به عنوان دسته غیرنرمال و مابقی رکوردها به عنوان دسته نرمال برچسب زده می‌شوند. این برچسب همان فیلد class از بسته داده‌ای شکل 5 را تشکیل می‌دهد.

در آخرین گام پردازش رکوردهای رویدادنگاری، فیلد پرچم را به صورت دودوئی و در حقیقت به همان صورتی که توسط سیستم‌عامل مورد پردازش قرار می‌گیرد، تجزیه می‌کنیم. از این رو یک فیلد واحد با مجموعه‌ای از فیلدها که هر یک حاوی صفر یا یک هستند، جایگزین می‌شود. از طرف دیگر فیلدهای مد و طول بافر بکار رفته از حالت هگزادسیمال به حالت دسیمال تبدیل می‌شوند و جهت نرمال‌سازی بر بزرگترین مقدار موجود برای این ویژگی‌ها تقسیم می‌شوند. عمل مشابهی نیز در مورد بر چسب‌ها انجام می‌شود و جهت نرمال‌سازی برچسب‌ها بر بزرگترین برچسب اختصاص‌داده‌شده تقسیم می‌شوند. نرمال‌سازی باعث می‌شود که مقادیر عددی در بازه صفر و یک قرار گیرند. این عمل جهت یادگیری بهتر شبکه بیزین و جلوگیری از بروز خطا در سیستم ضروری است.

3-4- دسته‌بندی بیزین

شبکه بیزین یک گراف جهت‌دار و بدون دور (DAG) می‌باشد که هر گره یک متغیر دامنه‌ای و هر یال وابستگی احتمالی را نشان می‌دهند و برای هر نود یک توزیع احتمالی شرطی (جدول CP[1]) وجود دارد[7] . شبکه بیزین برای محاسبه احتمال شرطی یک نود با استفاده از مقادیر تخصیص داده شده به نودهای دیگر بکار می‌رود؛ از این رو یک شبکه بیزین به عنوان یک دسته‌بند، توزیع احتمالی پسین نود کلاسی را با استفاده از مقادیر دیگر صفات تولید می‌کند. سودمندی اصلی شبکه‌های بیزین نسبت به دیگر انواع الگوهای پیشگویانه از قبیل شبکه‌های عصبی آن است که ساختار شبکه بیزین روابط ما بین ویژگی‌های مجموعه داده‌ای را بازنمایی می‌کند. به گونه‌ای انسان می‌تواند ساختارهای شبکه را درک کرده و در صورت لزوم آن‌ها را برای بدست آوردن الگوهای پیشگویانه بهتر اصلاح نمایند. دو کار اصلی در یادگیری یک شبکه بیزین عبارتند از: یادگیری ساختار گرافیکی و سپس آموزش پارامترها.

شبکه بیزین گروهی از روابط استقلال شرطی[2] مابین نودها را مطابق با مفهوم D-Separation[7]، کد می‌کند. این شیوه نگرش، یادگیری ساختار شبکه بیزین از طریق شناسایی روابط استقلال شرطی مابین نودها را پیشنهاد می‌کند. این الگوریتم‌ها با عنوان الگوریتم‌های مبتنی بر استقلال شرطی یا الگوریتم‌های مبتنی بر محدودیت شناخته می شوند[11][10].

برای دسته‌بندی گراف بیزین، از نرم‌افزار PowerPredictor [5] که دو الگوریتم General Bayesian Network و Multi Net را برای دسته‌بندی بکار می‌برد، استفاده می‌کنیم. دو الگوریتم بکار رفته برای یادگیری در این نرم‌افزار CBL1[10] (با فرض اینکه ترتیب نودها و در نتیجه جهت یال‌ها مشخص است) و CBL2 [12] (با فرض اینکه ترتیب نودها داده نشده است) می‌باشند. هر دو الگوریتم مبتنی بر روابط استقلال شرطی می‌باشند که از تئوری اطلاعات برای تحلیل وابستگی استفاده می‌کنند. هر دو الگوریتم از یک الگوریتم یادگیری سه‌فازی بهره می‌گیرند، این سه‌فاز عبارتند از: ایجاد طرح اولیه که بر مبنای الگوریتم ساخت درخت Chow-Liu می باشد، چگال‌سازی که یال‌هایی را به طرح اولیه می‌افزاید و سبک‌سازی که یال‌های غیرضروری را حذف می‌کند. این الگوریتم‌ها تضمین می‌کنند، هنگامی که الگوی اساسی داده‌ها فرضیات ویژه‌ای را برآورده می‌کند ساختار بهینه‌ای را یادگیری نمایند.

4- ارزیابی

برای تست سیستم از یک PC با پردازنده Core 2 Due GHz 2.4 با GB 1 رم و سیستم‌عامل لینوکس با هسته 2.6.30 استفاده شده است. پردازنده دارای تکنولوژی مجازی‌سازی می‌باشد.

[1] Conditional Probability

[2] Conditional Independancy

4-1- زمان اجرایی

برای سنجش زمان لازم برای هر فراخوان سیستم از برنامه strace استفاده شده که سربار حاصل از اجرای برنامه strace نیز به زمان حقیقی انجام هر فراخوان افزوده شده است. در این آزمایش، زمان انجام تعداد محدودی از فراخوان‌های سیستمی در سه حالت مورد بررسی قرار گرفته است: لینوکس بدون رویدادنگاری، لینوکس بر روی SCInterceptor به همراه رویدادنگاری و لینوکس به همراه دایمون رویدادنگاری[1]. هر یک از فراخوان‌های سیستمی 1000 بار فراخوانی و متوسط زمان فراخوانی‌ها محاسبه شده است(جدول 1).

جدول 1: زمان لازم برای اجرای فراخوان‌های سیستمی به میکرو ثانیه برای 1000 بار اجرای هر فراخوان

System Call	Linux	Linux + SCInterceptor	Audit Daemon
getpid	8	16	12
getuid	9	18	14
getgid	8	19	13
open	17	45	37
read	15	34	29
write	41	105	94
close	14	22	18
mkdir	31	75	95
rmdir	24	47	39

4-2- مقدار حافظه مورد نیاز

تنها مرحله‌ای که نیازمند تخصیص مقداری حافظه از فضای آدرسی کرنل است، مرحله تزریق کد در حافظه می‌باشد که برای وساطت فراخوان‌های سیستمی تزریق می‌شود. تعداد کل فراخوان‌های سیستمی وساطت شده 30 عدد می‌باشد و فراخوان‌هایی را شامل می‌شود که بیشتر در ارتباط با فایل سیستم هستند و در بروز حملات نیز بیشتر درگیر می‌باشند.

به طور تقریبی برای هر تزریق کد 3 کیلو بایت از حافظه لازم است که در مجموع 90=3*30 کیلو بایت از فضای آدرسی کرنل را اشغال می‌کند. از این رو سربار حافظه ای معماری ارائه شده بسیار اندک و ناچیز می‌باشد.

4-3-کارآیی الگو

در سیستم عامل لینوکس در مجموع 62 فراخوان سیستمی در ارتباط با فایل سیستم وجود دارد. از این میان ما تعداد 30 فراخوان سیستمی (مانند read, write, create, chmod, mkdir, link, ioctl, ...) را که برای انجام عملیات تشخیص نفوذ، کارآیی بیشتری دارند را انتخاب و به شیوه ای که عنوان شد، فراخوانی آن‌ها را رویدادنگاری کردیم.

برای آزمون میزان کارآیی الگوی پیشنهادی جهت تشخیص نفوذ، آزمونی را در دو مرحله پی‌ریزی می‌کنیم. در مرحله اول تشخیص نفوذ را بدون ساخت درخت هافمن انجام داده و در مرحله بعد همین آزمون را با ساخت درخت هافمن تکرار می‌کنیم.

برای شروع آزمون و یادگیری دسته‌بندی کننده شبکه بیزین مورد استفاده از کرم Adore و روتکیت‌های Gold2 و ShKit استفاده شده است. بعد از نصب این بدافزارها بر روی کامپیوتر به شیوه‌ای که قبلا مطرح کرده‌ایم، به رویدادنگاری فراخوان‌های سیستمی موردنظر خود پرداختیم.

جدول 2: رویدادنگاری انجام شده برای آموزش دسته بندی کننده شبکه بیزین

Rootkit/Worm	Adore&Gold2&ShKit
تعداد رکوردهای ثبت شده	1019893
تعداد رکوردها بعد از ساخت درخت هافمن	18037
تعداد اعضای مجموعه آموزشی برای داده‌های خام	60000
زمان لازم برای یادگیری با داده‌های خام بر حسب دقیقه	44
تعداد اعضای مجموعه آموزشی برای داده‌های پردازش شده با الگوریتم هافمن	18037
زمان لازم برای یادگیری با داده‌های پردازش شده با الگوریتم هافمن	25
دقت تشخیص نفوذ برای شبکه آموزش دیده با داده‌های خام	90/4%
دقت تشخیص نفوذ برای شبکه آموزش دیده با داده‌های بدست آمده از درخت هافمن	95/2%

جهت شناسایی PID پردازه‌های مخرب از نرم‌فزار chkrootkit استفاده کرده‌ایم (فقط برای فاز یادگیری این عمل انجام می شود). رویدادنگاری 5 ساعت به طول انجامید و در طول این مدت 1019893 رکود رویدادنگاری گردید. بعد از اتمام عملیات رویدادنگاری آنچه را که ثبت شد توسط برنامه تحلیل‌گر نوشته شده مورد تجزیه و تحلیل قرار می‌دهیم. این عمل به این دلیل انجام می‌شود تا هم جهت آموزش و یادگیری دسته‌بندی کننده شبکه بیزین و با کمک نرم‌افزار chkrootkit برچسب کلاسی رکوردهای مجموعه داده‌ای آموزشی را مشخص کنیم و هم اینکه بسته‌های داده‌ای مورد نظر را ایجاد نماییم.

جدول 3: دقت الگوریتم بکار رفته برای تشخیص نفوذ با داده های خام

[1] Audit Daemon

RootKit/Worm	ShitC Worm	Omega Worm	Suckit rootkit	Volc rootkit
تعداد رکوردهای ثبت شده در مدت 3 ساعت	432185	468201	445232	473450
تعداد رکورد های به کار رفته جهت تشخیص نفوذ	9000	9000	9000	9000
دقت تشخیص نفوذ برای شبکه آموزش دیده با داده های خام	86/1%	87/4%	88/01%	85/2%

جدول 4: دقت الگوریتم بکار رفته برای تشخیص نفوذ با داده های ساختار یافته

RootKit/Worm	ShitC Worm	Omega Worm	Suckit rootkit	Volc rootkit
تعداد رکوردهای ثبت شده در مدت 3 ساعت	432185	468201	445232	473450
تعداد رکورد های حاصله از درخت هافمن	9012	9852	1045	9234
تعداد رکورد های به کار رفته جهت تشخیص نفوذ	9000	9000	9000	9000
دقت تشخیص نفوذ برای شبکه آموزش دیده با داده های بدست آمده از درخت هافمن	93/2%	90/1%	92/2%	89/1%

عملیات رویدادنگاری برای چند کرم و روتکیت دیگر نیز تکرار شد، اما در مورد آنها دیگر نیازی به تعیین برچسب کلاسی نیست و این عمل توسط دسته‌بندی کننده مورد استفاده، انجام می‌شود. نتایج کار در جداول2و3و4 قابل مشاهد است. در مرحله یادگیری نسبت داده‌های مورد استفاده جهت یادگیری به داده بکار رفته جهت آزمون داخلی 33 به 67 می‌باشد.

5- نتیجه‌گیری

با توجه به گستردگی روز افزون تهدیدات بدافزارها و تلاش آن ها برای بی اثر کردن سامانه‌های تشخیص نفوذ، اخیرا عمده تلاش محققان سعی بر استفاده از مانیتورهای ماشین مجازی برای ایجاد سامانه‌های تشخیص نفوذ بوده است.

در این مقاله مانیتور ماشین مجازی SCInterceptor که مبتنی بر Bitvisor می باشد را ارائه دادیم که با استفاده از تکنولوژی مجازی‌سازی مبتنی بر سخت‌افزار روشی با کارایی بالا برای تشخیص نفوذ در سیستم عامل دارد و توسط محدودیت‌های پردازنده از حملات لایه‌های بالا نیز در امان خواهد بود. علاوه بر این کوچک بودن کد ناظر اطمینان بیشتری در مورد صحت عملکرد سیستم می‌دهد[19].

به علاوه، به دلیل انبوه رکوردهای رویدادنگاری شده سیستم‌عامل در یک بازه زمانی مشخص و وجود افزونه اطلاعاتی در این رکوردها، سامانه‌ای با استفاده از درخت هافمن مورد استفاده قرار دادیم که علاوه بر حفظ ماهیت کلی داده‌ها ساختار مناسبی برای تشخیص نفوذ ارائه می‌دهد.

با توجه به شرایط عدم قطعیت داده‌های رویدادنگاری شده، در گام بعدی با استفاده از دسته‌بند شبکه‌های بیزین بسته‌های داده‌ای ایجاد شده توسط درخت هافمن را دسته بندی کردیم و توانستیم نتایج خوبی را در مورد نفوذهای رخ داده در سیستم بدست آوریم. ارائه ایده استفاده از روش های فشرده‌سازی رکوردهای ورودی در یک الگوریتم تصمیم‌گیری چون شبکه‌های بیزین از مزایای معماری پیشنهادی توسط ما می‌باشد.

مراجع

[1] T. Garfinkel. "Traps and Pitfalls: Practical Problems in System Call Interposition Based Security Tools". In Proceedings of the 10th Annual Network and Distributed Systems Security Symposium, San Diego, February 2003.

[2]Koichi_Onoue, Yoshihiro Oyama, and Akinori Yonezawa. "Control of System Calls from Outside of Virtual Machines".In Proceedings of the 23rd Annual ACM Symposium on Applied Computing, March 2008:pp. 2116-1221.

[3]Xuxian Jiang, Xinyuan Wang, "'Out-of-the-Box' Monitoring of VM-Based High-Interaction Honeypots," Proceedings of the 10th International Symposium on Recent Advances in Intrusion Detection (RAID 2007), Queensland, Australia, September 2007:pp. 128-138.

[4] Takahiro Shinagawa, Hideki Eiraku, Kouichi Tanimoto, Kazumasa Omote, Shoichi Hasegawa, Takashi Horie, Manabu Hirano, Kenichi Kourai, Yoshihiro Oyama, Eiji Kawai, Kenji Kono, Shigeru Chiba, Yasushi Shinjo, Kazuhiko Kato: "BitVisor: a thin hypervisor for enforcing i/o device security". VEE 2009: pp.121-130.

[5] Cheng,J. PowerPredictor System: http://www.cs.ualberta.ca/~jcheng/bnpp.htm, 2000.

[6] Ashlesha Joshi, Samuel T. King, George W. Dunlap, Peter M. Chen. "Detecting Past and Present Intrusions Through Vulnerability-Specific Predicates". in Proc. 20th ACM SOSP, 2005: pp. 91-104.

[7]Pearl, J. "Probabilistic Reasoning in Intelligent Systems: networks of plausible inference". Morgan Kaufmann Publishers Inc, 1988, Pages: 552 .

[8]Cooper, G.F. and Herskovits, E. "A Bayesian Method for the induction of probabilistic networks from data". Machine Learning, 9, 1992: pp. 309-347.

[9] Heckerman, D. "A tutorial on learning Bayesian networks". Technical Report MSR-TR. Microsoft Research, 1995: pp. 95-06.

[10] Cheng, J., Bell, D.A. and Liu, W. "An algorithm for Bayesian belief network construction from data". In Proceedings of AI &STAT, Florida, 1997a: pp.83-90.

[11] Spirtes, P., Glymour, C. and Scheines, R. "Causation, Prediction, and Search" http://hss.cmu.edu/html/departments/philosophy/TETRAD.BOOK/book.html, 1993.

[12] Jie Cheng, David A. Bell, Weiru Liu: "Learning Belief Networks from Data: An Information Theory Based Approach". In Proceedings of ACM CIKM 1997: pp. 325-331.

[13] Jie Cheng, Russell Greiner. "Learning Bayesian Belief Network Classifiers: Algorithms and System". proceedings of the fourteenth Canadian conference on artificial intelligence, 2001(AI'2001): pp.141-151.

[14] Linux kernel Audit Daemon: http:// linux.die.net/man/8/auditd.

[15] Strace, Linux Debugging utility: http://linux.die.net/man/1/strace

[16] Anil Somayaji,"Operating System Stability and Security through Process Homeostasis", PhD. thesis, The University of New Mexico, July 2002.

[17 Huffman's original article: D.A. Huffman, "A Method for the Construction of Minimum-Redundancy Codes", Proceedings of the I.R.E., September 1952: pp 1098–1102

[18] Qemu Virtual Machine Monitor: www.qemu.org

[19]Vinod Ganapathy, Matthew J. Renzelmann, Arini Balakrishnan, Michael M. Swift, Somesh Jha: The design and implementation of microdrivers. ASPLOS 2008: pp168-178

[20] The User-mode Linux Kernel Home Page. http://user-mode-linux.sourceforge.net.