****

دانشگاه آزاد اسلامی

واحد زنجان

دانشکده برق، کامپیوتر و فناوری اطلاعات

**پایان‌نامه برای دریافت درجه کارشناسی ارشد «M.A»**

رشته: مهندسی کامپیوتر - گرایش نرم‌افزار

عنوان:

**محافظت از هسته سیستم عامل در برابر بد افزارها با استفاده از تکنیک مجازی سازی سخت افزاری**

استاد راهنما:

**دکتر علی اصغر خواصی**

نگارش:

**محمد ابراهیم زارع**

**پاییز 1395**



**فهرست مطالب**

[1 - چکیده 6](#_Toc465727033)

[2 - مقدمه 7](#_Toc465727034)

[2.1- معرفی 7](#_Toc465727035)

[2.2- بیان مسئله 8](#_Toc465727036)

[2.3 - پرسش اصلی تحقیق 10](#_Toc465727037)

[2.4 - نوآوری 10](#_Toc465727038)

[2.5 - اهمیت و ضرورت انجام تحقیق 10](#_Toc465727039)

[2.6 - مرور ادبیات و سوابق مربوطه 11](#_Toc465727040)

[3 - معرفی معماری سیستم‌عامل 13](#_Toc465727041)

[3.1 - طرح کلی 13](#_Toc465727042)

[3.2 - مدیریت حافظه در پردازنده‌های x86-386 و amd64 14](#_Toc465727043)

[3.3 - مکانیزم ترجمه 15](#_Toc465727044)

[3.3.1 - جداول توصیفگر 17](#_Toc465727045)

[3.3.2 - گزینشگرها 18](#_Toc465727046)

[3.3.3 - ثبات‌های قطعه 19](#_Toc465727047)

[3.3.4 - ترجمه به صفحه 20](#_Toc465727048)

[3.3.5 - جداول صفحه 21](#_Toc465727049)

[3.3.6 - مدخل‌های جدول صفحه 21](#_Toc465727050)

[3.3.7 - حافظه میانگیر ترجمه صفحه 23](#_Toc465727051)

[3.4 - حفاظت 23](#_Toc465727052)

[3.4.1 - خلاصه‌ای از مکانیزم‌های حفاظت 25](#_Toc465727053)

[3.4.2 - حفاظت در سطح قطعات حافظه 25](#_Toc465727054)

[3.4.3 - بررسی نوع 26](#_Toc465727055)

[3.4.4 - بررسی محدوده 29](#_Toc465727056)

[3.4.5 - سطح دسترسی 30](#_Toc465727057)

[3.4.6 - محدودیت در دسترسی به داده 31](#_Toc465727058)

[3.4.7 - محدودیت در انتقال کنترل 32](#_Toc465727059)

[3.4.8 - جایگزینی پشته 37](#_Toc465727060)

[3.4.9 - بازگشت از روال 38](#_Toc465727061)

[3.4.10 - اجرای فراخوانی سریع به‌واسطه دستورات 39](#_Toc465727062)

[3.4.11 - دستورات ویژه 40](#_Toc465727063)

[3.4.12 - دستورات حساس 41](#_Toc465727064)

[3.4.13 - حفاظت در سطح صفحه 41](#_Toc465727065)

[4 - و نوع از حفاظت در صفحه به کار گرفته می‌شود. 41](#_Toc465727066)

[5 -معماری سیستم‌عامل 44](#_Toc465727067)

[5.1 - معماری سیستم عامل ویندوز 44](#_Toc465727068)

[5.2 - روند یک فراخوانی سیستم 47](#_Toc465727069)

[5.3 - راه‌انداز هسته 48](#_Toc465727070)

[6 - روت‌کیت 50](#_Toc465727071)

[6.1 - روال محافظتی در برابر روت‌کیت‌ها 51](#_Toc465727072)

[6.1.1 - روش‌های جلوگیری از نصب و برگزاری ماژول غیر معتبر در هسته سیستم‌عامل 51](#_Toc465727073)

[6.1.2 - روش‌های شناسایی رفتار و تغییرات مخرب در فضای هسته سیستم‌عامل 52](#_Toc465727074)

[6.2 - روش‌های مورداستفاده روت‌کیت‌ها در دور زدن مکانیزم‌های امنیتی 53](#_Toc465727075)

[6.2.1 - افزایش دسترسی 53](#_Toc465727076)

[6.2.2 - غیرفعال سازی روش‌های اعتبار سنجی ماژول‌های هسته 53](#_Toc465727077)

[6.2.3 - غیرفعال سازی روال‌های بررسی دوره‌ای هسته 54](#_Toc465727078)

[6.3 - استفاده مجازی‌سازی در غیرفعال سازی روت‌کیت‌ها 55](#_Toc465727079)

[7 - معرفی مکانیزم ماشین مجازی در پردازنده‌ها 58](#_Toc465727080)

[7.1.1 - تاریخچه 58](#_Toc465727081)

[7.1.2 - معماری ماشین مجازی 59](#_Toc465727082)

[7.2 - چرخه فعالیت برنامه VMM 60](#_Toc465727083)

[7.3 - معرفی اجمالی ساختار vmcs 61](#_Toc465727084)

[7.4 - ساختمان داده VMCS 61](#_Toc465727085)

[8 -کارهای انجام‌شده در به‌کارگیری مجازی در امنیت 64](#_Toc465727086)

[9 - معرفی روش 69](#_Toc465727087)

[9.1 - روش‌های محافظت از مکانیزم‌های امنیتی 69](#_Toc465727088)

[9.2 - نحوه پیدا کردن محل حساس روال‌های امنیتی 70](#_Toc465727089)

[9.3 - نحوه تعریف نقطه شکست سخت‌افزاری 70](#_Toc465727090)

[9.4 - الگوریتم 71](#_Toc465727091)

[9.5 - ملاحظات الگوریتم پیشنهادی 72](#_Toc465727092)

[10 - - بررسی میزان اثرگذاری بر روی کارایی سیستم 72](#_Toc465727093)

[11 - نتیجه‌گیری 73](#_Toc465727094)

[منابع 74](#_Toc465727095)

**فهرست شکل‌ها**

[شکل شماره 1 16](#_Toc465891901)

[شکل شماره 2- ساختار یک توصیفگر 17](#_Toc465891902)

[شکل شماره 3- نحوه تبدیل آدرس منطقی به آدرس خطی 18](#_Toc465891903)

[شکل شماره 4 - ساختار گزینشگر 19](#_Toc465891904)

[شکل شماره 5- ساختار آدرس خطی 20](#_Toc465891905)

[شکل شماره 6- نحوه ترجمه از آدرس خطی 21](#_Toc465891906)

[شکل شماره 7- ساختار مدخل صفحه 22](#_Toc465891907)

[شکل شماره 8 - وضعیت رجیستر CR0 24](#_Toc465891908)

[شکل شماره 9 - فیلدهای حفاظتی در انواع توصیفگرها 27](#_Toc465891909)

[شکل شماره 10 - حلقه‌های حفاظت 31](#_Toc465891910)

[شکل شماره 11- روند بررسی سطح دسترسی به قطعات داده 33](#_Toc465891911)

[شکل شماره 12- بررسی اجازه دسترسی در انتقال کنترل 35](#_Toc465891912)

[شکل شماره 13- ساختار توصیفگر دروازه فراخوانی 36](#_Toc465891913)

[شکل شماره 14- نحوه بررسی اجازه دسترسی در انتقال کنترل با استفاده از دروازه 38](#_Toc465891914)

[شکل شماره 15- فیلدهای حفاظتی در صفحه 43](#_Toc465891915)

[شکل شماره 16- ساختار سیستم‌عامل ویندوز NT 46](#_Toc465891916)

[شکل شماره 17- روند اجرای یک API 49](#_Toc465891917)

[شکل شماره 18 74](#_Toc465891918)

# چکیده

جلوگیری از نفوذ به هسته سیستم‌عامل‌ها از مهم‌ترین مباحث امنیت در IT هست. روش‌های موجود از ضعف عدم ایجاد امنیتی به‌واسطه سخت‌افزار رنج می‌برند و نمی‌توانند امنیتی کامل و صد در صد را تضمین کنند. در این پژوهش سعی شده است با به‌کارگیری امکانات مجازی‌سازی که توسط پردازنده در اختیار قرار می‌گیرد یک‌لایه دسترسی جدید تعریف کنیم که از دسترس کل سیستم‌عامل حتی هسته نیز خارج باشد تا بتوان هسته را تحت نظر داشت بدون آنکه روال‌های نظارتی توسط بدافزارها تهدید بشوند.

**کلمات کلیدی : امنیت – هسته سیستم عامل – مجازی سازی سخت افزاری**

# مقدمه

## معرفی

هسته سیستم‌عامل‌ها جزء اهداف ایدئال برای نفوذ و آلوده کردن دستگاه‌های کامپیوتری هست. دسترسی به هسته امکان مخفی سازی خوبی به بدافزارها می‌دهد و همچنین امکان دور زدن تمامی مکانیزمامنیتی به‌کاررفته در سیستم را فراهم می‌سازد. از طرفی دسترسی به هسته به‌ناچار در سیستم‌عامل باید وجود داشته باشد تا برنامه‌های راه‌انداز سخت‌افزارها و یا برنامه‌های کنترلی مانند فایروال‌ها بتوانند در سیستم اجرا شوند. در تمامی سیستم‌عامل‌های امروزی دسترسی به هسته سیستم‌عامل با مکانیزم‌های سخت‌گیرانه‌ای کنترل می‌شود.

مکانیزم‌های استفاده‌شده در کنترل دسترسی به هسته سیستم‌عامل به علت آن‌که متکی بر اعتبار سنجی عمل می‌کنند و امکان جعل اعتبار هرچند بسیار مشکل و پیچیده وجود دارد، همیشه در معرض دور خوردن و ازکارافتادن قرار دارند. دسته بزرگی از بدافزارها را بدافزارهای ریشه[[1]](#footnote-1) تشکیل می‌دهند که بانفوذ به هسته، اعمال خرابکارانه خود را انجام می‌دهند. درنتیجه روش‌هایی توسعه داده‌شده‌اند تا به‌جای آن‌که جلوی نفوذ را بگیرند، تغییرات در هسته را تشخص داده و مانع آن می‌شوند.

در این پژوهش سعی شده است با استفاده از مکانیزم مجازی‌سازی سخت‌افزاری یک‌لایه امنیتی غیرقابل نفوذ ایجاد کرده و در آن روتین‌هایی تعبیه گردد تا از تغییرات در هسته جلوگیری کنند. این روش علاوه بر امنیتی بسیار خوبی که ایجاد می‌کند، حداقل سربار را هم بر سیستم می‌گذارد.

مجازی‌سازی سخت‌افزاری امکانی است که توسط پردازنده‌ها نسل جدید فراهم‌شده است. توسط این امکان می‌توان علاوه بر ۳ سطح دسترسی موجود در پردازنده سطح دیگر تعریف کرد که از دسترسی بیشتری حتی از سطح صفر برخوردار است. البته این سطح تنها برای کنترل و تغییر رفتار پیش‌فرض سیستم ایجادشده است که بتوان به‌وسیله آن ماشین مجازی برای روی سیستم ایجاد کرد. در این تحقیق از این لایه برای کنترل دسترسی به حافظه در سطح صفر استفاده می‌شود که مانع از رونوشت تصویر کدهای کرنل در حافظه گردد.

با استفاده از مجازی‌سازی سخت‌افزاری می‌توان بسیاری از دسترسی‌ها به سخت‌افزار را کنترل کرد. ازجمله این دسترسی‌ها می‌توان به کنترل تمام درگاه‌های ورودی و خروجی سخت‌افزاری و همچنین روال‌های دسترسی به تنظیمات و کنترل تغذیه می‌توان نام برد. ازجمله مواردی که می‌توان به‌واسطه مجاز سازی کنترل کرد و در این تحقیق بر روی آن تمرکز شده است کنترل دسترسی بر روی حافظه و همچنین کنترل دسترسی بر روی رجیسترهای خاص و کنترلی در پردازنده هست.

## بیان مسئله

سیستم‌عامل‌ها به شکل متناوب در حال به‌روز شدن و گسترش می‌باشند. این گسترش باعث پیچیده‌تر شدن آن‌ها نیز می‌گردد. پیچیدگی هر چه بیشتر سیستم‌عامل امکان ایجاد ضعف‌های امنیتی را در آن‌ها افزایش می‌دهد. این واقعیت تلاش‌هایی برای ایجاد امنیت و حفظ یک پارگی سیستم‌عامل را در برابر نفوذ توجیه می‌کند.

سیستم‌عامل‌های امروزی دارای دو بخش عمده عملکردی می‌باشند که شامل قسمت هسته و قسمت کاربردی می‌گردد. کدهای قسمت هسته با دسترسی ممتاز در پردازنده اجرا می‌شوند و قسمت کاربردی با دسترسی غیر ممتاز (سطح ۳) اجرا می‌گردند. به علت اینکه راه اندازه‌های سخت‌افزار و دیگر برنامه کاربردی مانند ضد بدافزارها و دیوارهای آتش شامل ماژول‌های می‌شوند که می‌باید همراه هسته سیستم‌عامل در سطح ممتاز اجرا شوند، امکان اجرای کد غیر از کد سیستم‌عامل با دسترسی ممتاز در سیستم‌عامل حتمی می‌باشد. از طرفی اجرای کد مخرب در هسته به علت دسترسی بدون محدودیت به منابع سیستم مانند حافظه و سخت‌افزار و همچنین دسترسی به ساختمان داده‌ها و کدها حیاتی هسته سیستم‌عامل برای بدافزارها بسیار جذاب هست.

همواره روش‌هایی که مانع اجرای کد مخرب در هسته سیستم‌عامل شود در طول زمان در سیستم‌عامل‌ها به کار گرفته‌شده است و بدافزارها نیز همواره از روش‌های گوناگون برای اجرای کد در هسته سیستم‌عامل استفاده می‌کرده‌اند.

روش‌هایی که سیستم‌عامل‌ها برای عدم اجرا کد مخرب استفاده می‌کرده و می‌کنند به شرح زیر هست.

دستورات لازم برای نصب و تغییر تنظیمات ماژول هسته تنها در فضای کاربری با دسترسی مدیر قابل انجام هست.

این مکانیزم از ابتدایی‌ترین سیاستی‌هایی امنیتی است که در سیستم‌عامل‌هایی مانند ویندوزهای مبتنی بر NT و لینوکس پیاده‌سازی شده است. این مکانیزم تنها از دسترسی بدافزارها در محیط‌های کاربری غیر مدیر جلوگیری می‌کند؛ و روال امنیتی برای محیط کاربری مدیر ندارد و اگر بدافزار در این محیط اجرا شود به کرنل دسترسی پیدا می‌کند. در این حالت آسیب‌پذیری‌های ارتقای دسترسی[[2]](#footnote-2) اگر در سیستم وجود داشته باشد منجر می‌شود که هسته سیستم نیز دچار مخاطره شود.

صحت سنجی ماژول‌هایی که در هسته اجرا می‌شوند به‌وسیله امضای دیجیتال معتبر ، این روش نیز توسط آسیب‌پذیری‌ها و نقطه‌ضعف‌هایی که در روش‌های امنیتی وجود دارد دچار خدشه شده است. به‌عنوان‌مثال به پروژه‌ای که شهریار جلایری [1] توسعه داده می‌توان اشاره کرد که به‌وسیله یک آسیب‌پذیری در یک ماژول معتبر، کدهای اعتبار سنجی ماژول را غیرفعال کرده در ادامه می‌تواند ماژول‌های هسته را بدون داشتن امضای معتبر در حافظه برگزاری کند.

روش‌هایی که جامعیت سیستم‌عامل را به شکل دوره‌ای بررسی می‌کنند و تغییرات در ساختمان داده و کدهای سیستم‌عامل را تشخیص می‌دهند. این روش‌ها نیز به علت اینکه حافظه کد روتین بررسی‌کننده در دسترس کدهای مخرب نفوذ کرده به هسته قرار دارند، به‌وسیله کدهای مخرب رونوشت می‌شوند و از کار می‌افتند. Skape [2] به‌طور کامل روال Pachgaurd (نامی است که شرکت مایکروسافت برای این مکانیزم بر روی سیستم‌عامل ویندوز قرار داده است) را توضیح داده است و روش غیرفعال کردن آن را نیز به شکل عملیاتی توضیح می‌دهد. با توجه به کار Skape می‌توان متوجه این واقعیت شد که روال‌های امنیتی Pachgaurd با پیچیدگی بسیار زیادی ایجادشده است و از روش‌های درهم‌ریختگی کد به شکل گسترده استفاده می‌کند ولیکن نهایتاً باوجوداین پیچیدگی‌ها این روال دور می‌خورد از کار می‌افتد.

دو روش آخر در سیستم‌عامل Windows از نسخه vista به بعد برای معماری 64 بیتی مورداستفاده قرار می‌گیرد.

از زمانی که امکان مجازی‌سازی سخت‌افزاری توسط پردازنده‌های شرکت Intel و AMD به وجود آمد روش‌های محافظتی از هسته سیستم‌عامل‌ها با استفاده از این امکان نیز توسعه پیدا کردند. با مجازی‌سازی می‌توان یک‌لایه امنیتی با دسترسی بیشتر از هسته ایجاد کرد که برای هسته به‌صورت کامل شفاف می‌باشد و می‌توان هرگونه دسترسی به حافظه و سخت‌افزار را به‌واسطه آن کنترل کرد. در این حالت روال‌های کنترل و امنیتی برای هسته و سیستم‌عامل و برنامه اجراشده در آن غیرقابل دسترسی هست درنتیجه امکان رونوشت کردن بر روی حافظه آن حتی از طریق هسته وجود ندارد. به‌وسیله اعمال مکانیزم امنیتی بر روی قسمت‌هایی حساس سیستم عامل می‌توان هرگونه تغییر بر روی هسته و خود روال امنیتی ماشین مجازی را کنترل و بلوک کرد و درنتیجه امنیت کامل با میزان اطمینان بسیار زیاد بر روی سیستم ایجاد کرد.

## پرسش اصلی تحقیق

آیا می‌شود با استفاده از مجازی‌سازی مانع ایجاد تغییرات در یکپارچگی کرنل سیستم‌عامل توسط بدافزارها شد.

آیا هزینه محاسباتی در مجازی‌سازی توسط سخت‌افزار برای ایجاد امنیت در سیستم قابل توجیه است.

## نوآوری

درروش مدنظر تنها نقاط حساس سیستم‌عامل هدف هست. ایده اصلی در این روش این است که اگر بشود تنها از مکانیزم‌های امنیتی موجود در هسته به‌درستی محافظت شود کافی است تا امنیت و یکپارچگی کل سیستم‌عامل را تأمین شود و همچنین هزینه انجام آن نیز بسیار پایین‌تر از محافظت از کل سیستم‌عامل می‌باشد.

## مشخصات نرم افزاری و سخت افزاری مورد تحقیق

در این پروژه سیستم‌عامل‌های ویندوز مبتنی بر NT به‌عنوان سیستم هدف جهت بررسی انتخاب‌شده است و معماری پردازنده نیز x86 در نظر گرفته شده است. البته آزمایشات و پیاده سازی ها به شکل خاص بر روی پردازنده های شرکت INTEL صورت گرفته ولیکن مکانیزم های بررسی شده به شکل کاملا مشابه در پردازنده های شرکت AMD موجود دارد.

## ساختار فصل‌ها

در فصل دو به مرور ادبیات تحقیق پرداخته شده است در این فصل دو بخش به سیستم حافظه مجازی و مکانیزم حافظت در پرداخته میشود و در ادامه ساختار سیستم عامل به شکل اجمالی توضیح داده شده است و در انتها مکانیرم مجازی سازی سخت افزاری به تنفضیل شرح داده خواهد شد.

در فصل سوم در مورد روت کیت ها و انواع روش های نفوذ به هسته که توسط آنها مورد استفاده قرار میگیرد توضیحاتی داده شده است و در ادامه در مورد روش های محافظتی به کار رفته در سیستم عامل برای جلوگیری از نفو‌ذ روت کیت ها به سیستم بحث شده است.

در فصل چهارم به معرفی روش ابدایی و الگوریتم کار پرداخته میشود و در فصل پنجم نیز میزان اثر گذاری بر روی کاررایی سیستم به وسیله نرم افزارهای محک پرداخته میشود.

# مرور ادبیات تحقیق

به علت اینکه سیاست گذاری بر روی دسترسی به حافظه مهمترین هدف این تحقیق است، ابتدا به معرفی مکانیزم مدیریت حافظه در پردازنده پرداخته میشود در ادامه مکانیزم حفاظت و مدیریت دسترسی به حافظه مورد بررسی قرار میگیرد. در قسمت دیگر به بررسی نحوه قرار گیری قسمت های مختلف سیستم عامل در حافظه و دسترسی های هر قسمت و هنچنین نحوه ارتباط بین قسمت ها پرداخته می‌شود. در انتها نیز به معرفی مکانیزم مجازی سازی سخت افزاری پرداخته شده است.

## طرح کلی حافظه

یکی از وظایف سیستم‌عامل در اختیار قرار دادن امکانات سخت‌افزار برای نرم‌افزارهای کاربردی ایست به‌نحوی‌که این برنامه‌ها بتوانند به‌طور مشترک از آن‌ها استفاده کنند. یکی از این امکانات که سخت‌افزار و به‌طور مشخص پردازنده در اختیار قرار می‌دهد بخش‌بندی حافظه بنا بر دسترسی کدهای در حال اجرا به آن‌ها است. در این حالت می‌توان حافظه را به بخش‌هایی تقسیم کرد و برای هر بخش دسترسی تعیین کرد. به‌نحوی‌که کدهای در حال اجرا تنها به نواحی مجاز دسترسی دارند.

در سیستم‌عامل حافظه ازنظر دسترسی به دو بخش کلی تقسیم می‌شود. سطح ممتاز و سطح کاربردی. کدهایی که از حافظه کاربردی اجرا می‌شوند تنها به حافظه کاربردی دسترسی دارند و از هرگونه دسترسی به حافظه ممتاز توسط پردازنده منع شده‌اند. کدهایی که از حافظه ممتاز اجرا می‌شوند هم به حافظه ممتاز دسترسی دارند و هم به حافظه کاربری.

قسمت‌هایی از سیستم‌عامل که اعمال بنیادی مانند مدیریت فایل‌سیستم، مدیریت شبکه، زمان‌بندی پردازش‌ها، راه‌انداز انواع سخت‌افزار را انجام می‌دهند در ناحیه ممتاز حافظه قرار می‌گیرد. به این قسمت‌ها کرنل سیستم‌عامل گفته می‌شود.

در ادامه برای روشن شدن کامل مفاهیم گفته‌شده مقداری در مورد مکانیزم‌های مدیریت حافظه در پردازنده بحث می‌شود.

## مدیریت حافظه در پردازنده‌های x86-386 و [[3]](#footnote-3)amd64

برای حفظ مطابقت با پردازنده‌های ابتدایی پردازنده‌ها در دو یا سه مد کاری مختلف کار می‌کنند. مدهای کاری پردازنده شامل موارد زیر می‌شود.

مد واقعی[[4]](#footnote-4): در این مد کاری پردازنده‌ها مطابق پردازنده 8086 عمل می‌کنند. در این مد خطوط آدرس‌دهی 20 بیتی است و حدود یک مگابایت قابل‌دسترسی هست و بقیه حافظه در صورت وجود غیرقابل استفاده است. برای آدرسی دهی از روش آدرس‌دهی ناحیه و فاصله استفاده می‌شود. در این حالت آدرس حافظه با جمع دو مقدار به دست می‌آید. یکی از آن‌ها یک ناحیه 64 کیلوبایتی را مشخص می‌کند و دیگری فاصله از ابتدای ناحیه را به دست می‌دهد. هیچ‌گونه مکانیزم نگاشت و ترجمه‌ای در آدرسی دهی وجود ندارد درنتیجه آدرس به‌دست‌آمده همان آدرس فیزیکی بر روی حافظه است. به همین دلیل به این مد کاری مد واقعی گفته می‌شود. سیستم‌عامل داس DOS از این نوع آدرس‌دهی استفاده می‌کرد.

مد محافظت‌شده[[5]](#footnote-5): این مد کاری برای اجرا پردازه‌های 32 بیتی طراحی گردیده است و از پردازنده 286 به بعد مورداستفاده قرارگرفته است. در این مد کاری تعداد خطوط آدرس‌دهی به 32 عدد افزایش پیداکرده است درنتیجه حدود 4 گیگابایت حافظه قابل آدرس‌دهی می‌باشند. در این مد آدرس‌ها به دودست آدرس منطقی و آدرس فیزیکی تقسیم‌بندی می‌شود. آدرس منطقی شامل آدرسی است در یک دستور ارجاع به حافظه در ثباتی و یا مقدار ثابت توسط قسمت دیکدر دستور به دست می‌آید. در این مرحله برای ارجاع به حافظه آدرس مشخص‌شده در دستور به‌وسیله ساختمان داده‌هایی که برای پردازنده مشخص‌شده به آدرس دیگری نگاشت و ترجمه می‌شود. آدرس به دست آماده را آدرس فیزیکی گویند. همین آدرس بر روی خطوط آدرسی دهی حافظه قرار می‌گیرد.

در این مد کاری می‌توان حافظه را به یک و یا بیش از یک ناحیه با طول تغییر تقسیم‌بندی کرد و برای هریک دسترسی و تنظیماتی خاصی در نظر گرفت. به نواحی حافظه قطعه[[6]](#footnote-6) گفته می‌شود. همچنین می‌توان مکانیزم را در پردازنده فعال کرد تا هر ناحیه را به زیر ناحیه کوچک‌تری البته با طول ثابت و مشخص تقسیم کرد و برای هر یک آن‌ها نیز دسترسی و تنظیماتی در نظر گرفت. به این زیر ناحیه‌ها صفحه[[7]](#footnote-7) گفته می‌شود.

در این مد اجرای بعضی از دستورات تنها در حالتی ممکن است که کد از ناحیه حافظه با دسترسی ممتاز (سطح صفر) خوانده‌شده باشد.

مد بلند[[8]](#footnote-8): این مد کاری برای اجرا در پردازنده‌های 64 بیتی طراحی‌شده است. طول آدرس منطقی در این مد 64 بیت هست. این مد کاری شباهت زیادی به مد محافظت‌شده دارد. عمده‌ترین تفاوت این مد با مد محافظت‌شده بیشتر شدن ثبات‌ها و همچنین حذف تعدادی از مکانیزم‌های ناحیه بندی کد که در سیستم‌عامل‌های امروزی مورداستفاده نبود، هست.

مکانیزم حفاظت و ترجمه آدرس منطقی دو مفهوم به‌هم‌پیوسته هستند که برای درک بهتر روند کار در پردازنده‌ها این دو مکانیزم به‌صورت دقیق در ادامه توضیح داده‌شده است.

## مکانیزم ترجمه

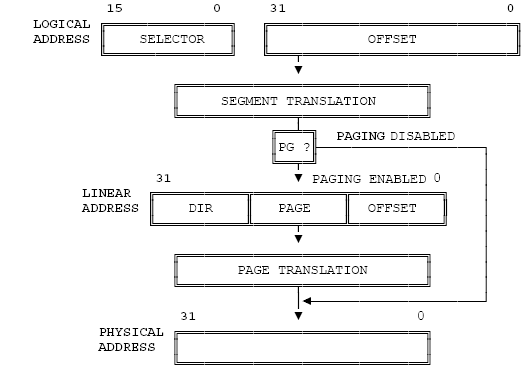
پردازنده تبدیل آدرس منطقی یا همان آدرسی که برای برنامه‌نویس دیده می‌شود به آدرس فیزیکی یا همان آدرسی که با آن به حافظه اصلی ارجاع می‌شود را در دو مرحله انجام می‌دهد.

* ترجمه به قطعه[[9]](#footnote-9): در این مرحله آدرس منطقی که شامل گزینشگر قطعه[[10]](#footnote-10) و فاصله از شروع قطعه است به آدرس خطی تبدیل می‌شود.
* ترجمه به صفحه[[11]](#footnote-11): در این مرحله آدرس خطی به آدرس فیزیکی تبدیل می‌گردد. این مرحله انتخابی است و به انتخاب طراح برنامه سیستمی می‌باشد.

فرآیند ترجمه به طریقی انجام می‌شود که برای برنامه در حال اجرا قابل‌دیدن نیست. شکل شماره 1 در یک دید کلی و تجریدی مراحل ترجمه یک آدرس را نشان می‌دهد.

برای انجام فرایند ترجمه آدرس، پردازنده به این ساختمان داده‌ها احتیاج دارد.

* توصیفگرها[[12]](#footnote-12)
* جداول توصیفگر[[13]](#footnote-13)
* گزینشگرها[[14]](#footnote-14)
* جداول گزینشگر

[[15]](#footnote-15)

شکل شماره 1

توصیفگر: توصیفگرهای قطعات حافظه، اطلاعات لازم برای نگاشت یک آدرس منطقی را به یک آدرس فیزیکی در اختیار قرار می‌دهند. توصیفگرها می‌توانند توسط کامپایلرها، لینکرها، بار کننده‌ها و یا سیستم‌عامل ایجاد شوند ولیکن توسط خود برنامه ایجاد نخواهند شد. ساختمان داده توصیفگرها شامل این چند فیلد می‌باشند:

فیلد پایه: محل شروع یک سگمنت که محدوده آدرس‌دهی تا 4 گیگابایت را مشخص می‌کند. طول این فیلد 32 بیت است و همان‌طور که در شکل شماره 2 مشخص‌شده شامل سه‌تکه جدا از هم می‌باشد.

فیلد محدوده: اندازه سگمنت را مشخص می‌کند. این فیلد در دوتکه جدا از هم به طول 20 بیت قرار گرفته‌شده است. پردازنده این فیلد را به دو نوع مختلف تفسیر می‌کند که بستگی به ست بودن یا نبودن بیت تکه‌ای[[16]](#footnote-16) از ساختمان داده توصیفگر می‌باشد. دو حالت موجود به‌قرار زیر است:

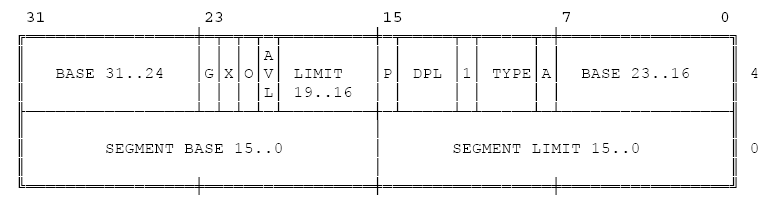
* بیت فیلد تکه‌ای توصیفگر واحدهای یک بایتی را مشخص کند که طبیعتاً محدوده‌ای به طول حداکثر 1 مگابایت را پوشش می‌دهد.
* بیت فیلد تکه‌ای توصیفگر واحده‌ای‌هایی 4 کیلوبایت را مشخص کند. در این حالت می‌توان محدوده‌ای تا 4 گیگابایت مشخص کرد.

بیت تکه‌ای: نحوه تفسیر فیلد محدوده را مشخص می‌کند. وقتی بیت ریست باشد در تفسیر محدوده، واحدهای آدرس 1 بایتی در نظر گرفته می‌شود؛ و در حالت ست واحدهای آدرس 4 کیلوبایتی تفسیر می‌گردد.

فیلد نوع: این فیلد نوع توصیفگر را مشخص می‌کند.

سطح دسترسی توصیفگر[[17]](#footnote-17): این فیلد در مکانیزم حفاظتی برای تعیین اجازه دسترسی به قطعه حافظه استفاده می‌شود که توضیح بیشتر را به قسمت بعد موکول می‌کنیم

بیت موجودیت قطعه: اگر این بیت صفر باشد محتوایی‌ات داخل توصیفگر برای ترجمه معتبر نیست. سیستم‌عامل‌هایی که حافظه مجازی را در سطح قطعه پیاده‌سازی می‌کنند می‌توانند از این فیلد برای تعیین معتبر بودن یا نبودن قطعه استفاده کنند.



شکل شماره 2- ساختار یک توصیفگر

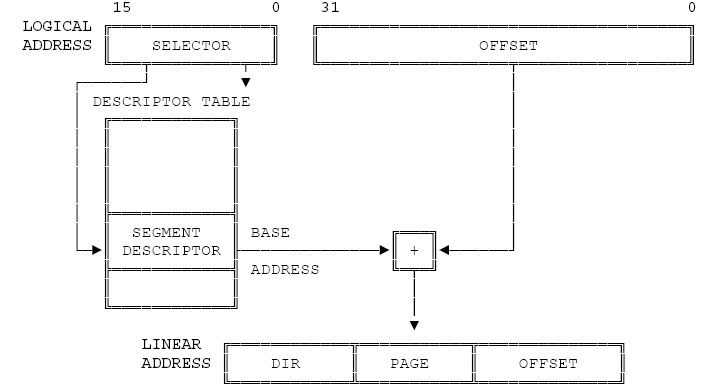
### جداول توصیفگر

توصیفگرهای سگمنت در دو نوع از جدول ذخیره می‌شود.

* جدول توصیفگر عمومی[[18]](#footnote-18)
* جدول توصیفگر محلی[[19]](#footnote-19)

جدول توصیفگر چیزی نیست جز یک آرایه ساده با مدخل‌های 8 بایتی که در هر مدخل یک توصیفگر قرارگرفته است. اندازه جدول تغییر است که حداکثر 2 به توان 13 توصیفگر در آن قرار می‌گیرد. اولین مدخل از جدول توصیفگر عمومی همیشه صفر است.

پردازنده این جداول را در حافظه قرار می‌دهد و آدرس شروع آن‌ها را در ثبات‌های GDTR (برای جدول عمومی) و LDTR (برای جدول محلی) قرار می‌دهد. دو دستور اسمبلی LGDT و SGDT برای خواندن و نوشتن مقدار در GDTR استفاده می‌گردد و همین‌طور دو دستور LLDT و SLDT برای کار با LDTR استفاده می‌گردد.



شکل شماره 3- نحوه تبدیل آدرس منطقی به آدرس خطی

### گزینشگرها

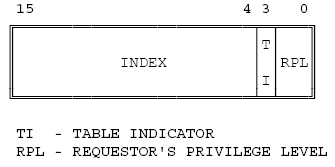
گزینشگرها قسمتی از آدرس منطقی است که برای مشخص کردن یک توصیفگر از جدول توصیفگر استفاده می‌شوند. گزینشگر می‌تواند توسط برنامه قابل‌دیدن باشد ولی معمولاً مقدار گزینشگر توسط لینکرها و یا بارگذارنده تعیین می‌شود. شامل فیلدهای زیر است.

فیلد شاخص: یکی از 1 تا 8192 تا توصیفگر موجود در جدول توصیفگر را مشخص می‌کند. پردازنده برای به دست آوردن آدرس توصیفگر، مقدار فیلد شاخص گزینشگر را در 8 ضرب کرده و شماره را با آدرس شروع جدول توصیفگر جمع می‌کند و آدرس توصیفگر را به دست می‌آورد.

فیلد مشخصه جدول[[20]](#footnote-20): این فیلد نوع جدول را مشخص می‌کند که اگر برابر با صفر باشد جدول از نوع GDT است و اگر برابر یک باشد از نوع LDT است.

فیلد سطح دسترسی درخواستی[[21]](#footnote-21): این فیلد در مکانیزم محافظتی استفاده می‌شود.

به دلیل آنکه اولین خانه از جدول GDT برای پردازنده بی‌مصرف است، گزینشگری که فیلد شاخص آن صفر باشد و همچنین فیلد مشخصه جدول نیز با صفر پرشده باشد (گزینشگر به اولین خانه از جدول GDT اشاره می‌کند) به‌عنوان گزینشگری با مقدار NULL بکار برده می‌شود. پردازنده در هنگام بارگذاری گزینشگری با مقدار NULL خطا ایجاد نکرده ولیکن در حالتی که با استفاده از گزینشگری با مقدار NULL به حافظه ارجاع شود تولید استثنا می‌کند. از این قابلیت برای به تله انداختن ارجاعات تصادفی استفاده می‌شود.



شکل شماره 4 - ساختار گزینشگر

### ثبات‌های قطعه

پردازنده اطلاعات توصیفگرها را در ثبات قطعه نگهداری می‌کند. با این کار مانع خواندن جدول توصیفگرها در هر بار ارجاع به حافظه می‌گردد.

هر ثبات قطعه از دو قسمت مرئی که توسط برنامه قابل‌دیدن است و قسمت نامرئی که توسط برنامه قابل‌دیدن نیست تشکیل می‌شود. برنامه با استفاده از دستورات زبان ماشین مقدار قسمت مرئی این ثبات‌ها را با یک گزینشگر 16 بیتی پر می‌کند؛ و پردازنده به‌طور خودکار آدرس پایه، محدوده سگمنت، نوع و دیگر اطلاعات موردنیاز را از توصیفگری که گزینشگر به آن اشاره می‌کند دریافت می‌کند و در قسمت نامرئی ثبات قرار می‌دهد.

برنامه به دو صورت می‌تواند مقدار ثبات‌های قطعه را تغییر دهد.

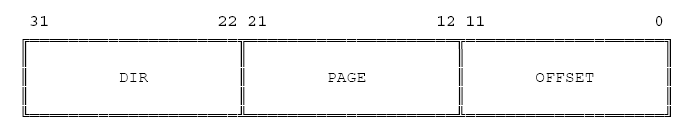
* با دستوراتی که به‌طور مستقیم مقدار ثبات را تغییر می‌دهد به‌طور مثال با دستورات MOV،POP،LDS،LSS،LGS،LFS این دستورات به‌طور روشن مقدار ثبات‌های قطعه را تغییر می‌دهد.
* با دستوراتی که به‌طور غیرمستقیم مقدار ثبات‌ها تغییر می‌دهند؛ مانند CALL، JUMP. این دستورات مقدار CS را به‌طور غیرمستقیم تغییر می‌دهد.

### ترجمه به صفحه

در مرحله دوم از ترجمه آدرس پردازنده آدرس خطی را به آدرس فیزیکال تبدیل می‌کند. در این مرحله خصوصیات پایه‌ای برای یک سیستم حافظه مجازی صفحه گرا را همراه با حفاظت در سطح صفحه پیاده‌سازی می‌کند.

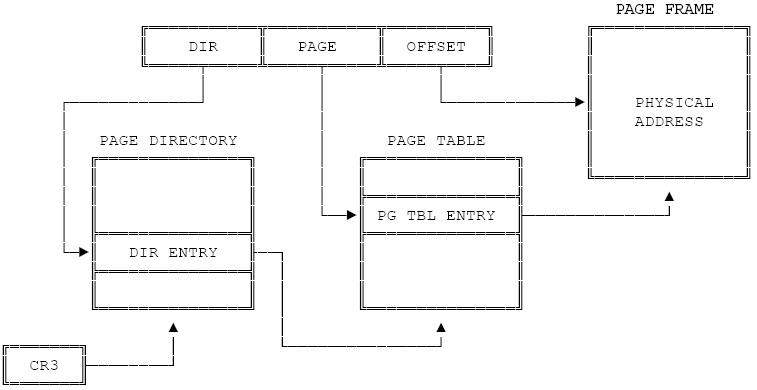
ترجمه به صفحه مرحله ایست انتخابی و برای فعال کردن آن می‌بایست بیت PG از رجیستر CR0 ست شده باشد. این بیت اغلب توسط سیستم‌عامل در حین مراحل بوت شدن ست می‌شود. اگر سیستم‌عامل از حافظه مجازی پشتیبانی می‌کند باید بیت PG ست شده باشد.

یک فریم صفحه یک‌تکه 4 کیلوبایتی از آدرس‌های هم‌جوار در حافظه فیزیکی است. آدرس خطی یک ارجاع غیرمستقیم به حافظه فیزیکی را با مشخص کردن یک جدول صفحه، یک صفحه و آفست از شروع صفحه صورت می‌دهد. ساختار آدرس خطی در شکل شماره 5 آمده است.



شکل شماره 5- ساختار آدرس خطی

در مکانیزم آدرس‌دهی فیلد DIR برای مشخص کردن مدخلی از دیکشنری صفحات و همین‌طور فیلد Page برای مشخص کردن مدخلی از جدول صفحات و OFFSET برای تعیین فاصله از آغاز صفحه به کار می‌رود. شکل شماره 6 نحوه ترجمه از آدرس خطی را به آدرس فیزیکی نشان می‌دهد.



شکل شماره 6- نحوه ترجمه از آدرس خطی

### جداول صفحه

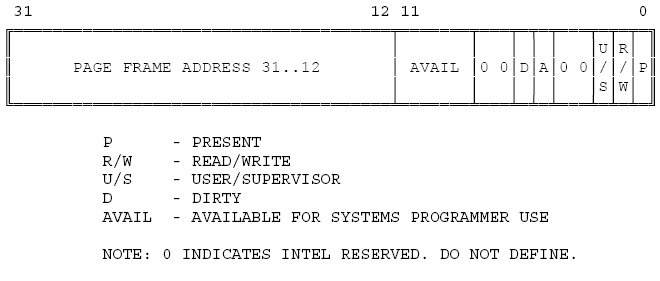
یک جدول صفحه آرایه‌ای ساده از مشخصات صفحه با اندازه 32 بیت است که خود نیز از یک صفحه تشکیل‌شده است. بنابراین 4 کیلوبایت از حافظه را اشغال می‌کند.

دو سطح از جداول در کنار هم یک صفحه را مشخص می‌کند. جدولی که در سطح اول قرارگرفته دیکشنری صفحات نام دارد که به 1024 جدول در سطح دوم اشاره می‌کند. به جدولی که در سطح دوم قرارگرفته جدول صفحه گفته می‌شود که به 1024 صفحه اشاره می‌کند. هر صفحه نیز خود 4 کیلوبایت فضا را پوشش می‌دهد پس در این حالت 1024\*1024\*4\*1024 بایت حافظه باقابلیت صفحه‌بندی قابل آدرس‌دهی می‌باشد.

آدرس فیزیکی دیکشنری صفحه جاری در داخل رجیستر CR3 نگهداری می‌شود. که به نام ثبات پایه دیکشنری صفحات خوانده می‌شود. نرم‌افزار مدیریت حافظه به‌صورت انتخابی قادر به استفاده از یک جدول صفحه مشترک برای همه وظیفه‌هاست است و هم استفاده از جدول صفحه متفاوت به ازای هر وظیفه می‌باشد.

### مدخل‌های جدول صفحه

مدخل‌های جدول صفحه در هر دو سطح دارای ساختاری مشابه هستند. شکل شماره 7 ساختار مدخل را نمایش می‌دهد



شکل شماره 7- ساختار مدخل صفحه

آدرس قاب صفحه: این فیلد آدرس فیزیکی یک صفحه را مشخص می‌کند. به علت اینکه صفحه محدوده 4 کیلوبایتی را پوشش می‌دهد. درنتیجه 12 بیت پایین آدرس قاب صفحه همیشه صفر است. در دیکشنری صفحات، آدرس قاب صفحه همان آدرس جدول صفحات است و در سطح دوم آدرس قاب صفحه شامل آدرس یک قاب صفحه است.

بیت حضور: این بیت نشان می‌دهد که آیا می‌توان از محتویات این مدخل برای کار ترجمه استفاده کرد. اگر برابر با یک باشد مدخل قابل‌استفاده است.

اگر بیت حضور برابر با صفر باشد، در زمان تلاش برای استفاده از مدخل جدول صفحه در عملیات ترجمه آدرس، پردازنده یک سیگنال استثنای صفحه ایجاد می‌کند. نرم‌افزار مدیریت حافظه مجازی از این استثنا پشتیبانی می‌کند به این صورت که با بروز استثنا صفحه خواسته‌شده به حافظه آورده می‌شود و دستوری که باعث ایجاد این استثنا شده نیز دوباره اجرا می‌شود.

بیت‌های دسترسی و دست‌خوردگی: این بیت‌ها اطلاعاتی در مورد صفحه استفاده‌شده در هر دو سطح جدول بیان می‌کند. این بیت توسط پردازنده ست می‌شود، ولیکن پردازنده این بیت‌ها را ری ست نمی‌کند.

پردازنده بیت دسترسی برای هر دو سطح جدول را قبل از عمل خواندن و نوشتن در صفحه ست می‌کند. همچنین پردازنده بیت دست‌خوردگی را قبل از عمل نوشتن در صفحه و تنها در سطح دوم جدول صفحه ست می‌کند، این بیت در دیکشنری جداول در سطح اول بلااستفاده می‌باشد.

سیستم‌عامل می‌تواند از این بیت برای تعیین اینکه کدام صفحه را در زمان احتیاج به تعویض صفحه از حافظه خارج کند استفاده کند.

بیت‌های خواندن یا نوشتن و همچنین بیت کاربر یا سرپرست: این بیت‌ها در عمل ترجمه شرکت نمی‌کنند بلکه در مبحث مکانیزم حفاظتی در سطح صفحه استفاده می‌شوند.

### حافظه میانگیر ترجمه صفحه

برای ایجاد ترجمه با کارایی مؤثر، پردازنده بیشترین جداول صفحه‌های استفاده‌شده را در حافظه میانگیر خود نگهداری می‌کند. تنها زمانی که جدول صفحه در حافظه میانگیر موجود نباشد به حافظه اصلی مراجعه می‌کند. حافظه مجازی برای برنامه کاربردی نامرئی است ولیکن برای برنامه سیستمی قابل‌مشاهده می‌باشد. برنامه نویسان سیستم‌عامل می‌بایست حافظه میانگیر را در زمان تغییر جدول صفحات تخلیه کنند. حافظه میانگیر را با دو روش می‌توان تخلیه کرد.

* با بارگذاری مجدد ثبات CR3 به‌وسیله دستور MOV، برای مثال MOV CR3، EAX
* در زمان سوییچ کردن به وظیفه دیگر اگر CR3 نیز تغییر کند حافظه میانگیر نیز تغییر پیدا می‌کند.

## حفاظت

پس از بررسی نحوه ترجمه آدرس حال به بررسی مکانیزم‌های حفاظتی در پردازنده می‌پردازیم.

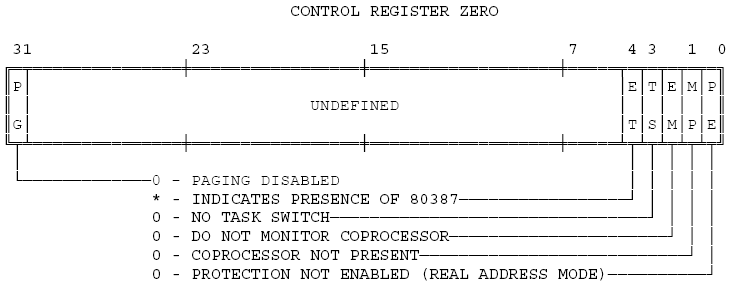
حالت آدرس‌دهی محافظت‌شده از امکانات گنجانده‌شده در پردازنده‌های 80286 به بعد است. در این حالت محدوده آدرس‌دهی تا 4 گیگابایت افزایش پیدا می‌کند و از آدرس‌دهی نسبی به آدرس‌دهی مستقیم تغییر میابد. برای استفاده از امکانات چند برنامگی می‌بایست از این حالت کاری استفاده کرد.

در زمان شروع به کار پردازنده وضعیت رجیسترها به شکل زیر است:

|  |  |  |
| --- | --- | --- |
| =00000002H | | EFLAGS |
| =0000FFF0H | | IP |
| =00H | | CS selector |
| =000H | | DS selector |
| =000H | | ES selector |
| =0000H | | SS selector |
| =0000H | | FS selector |
| =0000H | | GS selector |
| IDTR | | |
| =0 | base |  |
| =3FFH | limit |  |

جدول شماره 1 - مقادیر اولیه ثبات ها در زمان راه اندازی

و همچنین وضعیت رجیستر CR0 به حالت زیر است:



شکل شماره 8 - وضعیت رجیستر CR0

این تنظیمات دلالت بر این دارد که در زمان شروع به کار حالت آدرس‌دهی جاری، حالت واقعی می‌باشد و برنامه می‌بایست برای استفاده از قابلیت‌های گنجانده‌شده در حالت حفاظت‌شده چند ساختمان داده را قبل از تغییر حالت کاری ایجاد کند ازجمله جدول توصیفگرهای عمومی و محلی، جدول توصیفگر وقفه، پشته و در صورت استفاده از حافظه مجازی، جداول صفحه می‌باشد.

ازجمله مزایایی که در حالت حافظت شده قابل‌استفاده است، مکانیزم‌های حفاظتی می‌باشد که در بخش مدیریت حافظه به آن اشاره شد. هدف اصلی از مکانیزم‌های حفاظتی کمک به شناسایی و شناختن باگ‌ها می‌باشد. این‌گونه پردازنده‌ها از برنامه‌های پیچیده‌ای پشتیبانی می‌کند که ممکن از صد یا دویست ماژول مختلف تشکیل‌شده باشد. در این‌گونه از برنامه‌ها این سؤال مطرح است که چگونه می‌توان باگ‌ها را تشخیص و از برنامه زدود. برای کمک به اشکال‌زدایی سریع‌تر برنامه‌ها پردازنده دارای مکانیزم‌هایی هستند که برای بررسی و تأیید دسترسی به حافظه و متابعت اجرا کد با معیارهای حفاظتی استفاده می‌شود. این مکانیزم‌هایی می‌توانند بنا به انتخاب طراح سیستم استفاده‌شده و یا چشم‌پوشی شوند.

### خلاصه‌ای از مکانیزم‌های حفاظت

حفاظت دارای 5 جنبه می‌باشد

1. بررسی نوع توصیفگر
2. بررسی محدوده قطعات حافظه
3. محدود کردن در حوزه قابل آدرس‌دهی
4. محدود کردن در نقطه شروع روال‌ها
5. محدود کردن در مجموعه دستورات

حفاظت در هر دو بخش ترجمه به قطعه و ترجمه به صفحه اعمال می‌شود.

هر ارجاع به حافظه به‌وسیله سخت‌افزار بررسی‌شده تا صحت‌وسقم آن با ضوابط حفاظت تطابقت داده شود. همه این بررسی‌ها قبل از شروع سیکل حافظه انجام می‌شود. هرگونه تخلف مانع شروع سیکل حافظه می‌گردد و یک استثنا را موجب می‌شود.

اجازه دسترسی مفهومی است که در سه جزء 4،3 و 5 از حفاظت به‌عنوان بخش اصلی گنجانده‌شده است. در اعمال محدودیت به روال‌ها، اجازه دسترسی درجه‌ای از حفاظت است که مطمئن می‌سازد روال نمی‌تواند اشتباهی بکند که امکان تأثیر بر روی دیگر روال‌ها و داده‌ها را داشته باشد؛ و در اعمال به داده‌ها اجازه دسترسی درجه‌ای از حفاظت است که یک ساختمان داده می‌باید دارا باشد تا مطمئن شویم روال‌هایی مشخصی بر روی آن تأثیر می‌گذارند.

مفهوم اجازه دسترسی بر هردو حالت، محافظت در سطح قطعات حافظه و محافظت در سطح صفحات اعمال می‌شود.

### حفاظت در سطح قطعات حافظه

هر پنج جزء حفاظت بر روی قطعات حافظه اعمال می‌شوند. قطعه حافظه واحد حفاظت می‌باشد. توصیفگرهای قطعه شامل پارامتر حفاظتی قطعه‌ای می‌باشند که به آن اشاره و آن را توصیف می‌کنند. پارامترهای حفاظتی در زمان برگزاری توصیفگرها به داخل ثبات قطعه توسط پردازنده بررسی می‌شوند. در هر ثبات قطعه پارامترهای حفاظتی قطعه جاری آدرس‌دهی شده، نگهداری می‌شود. در شکل شماره 9 فیلدهای حفاظتی موجود در توصیفگر پررنگ شده است.

پارامترهای حفاظتی توسط برنامه‌های سیستمی (اغلب بخش مدیریت حافظه سیستم‌عامل) در زمان ایجاد توصیفگرها در آن‌ها گنجانده می‌شوند ولی برنامه‌های کاربردی احتیاجی به نگرانی در مورد پارامترهای محافظتی ندارند.

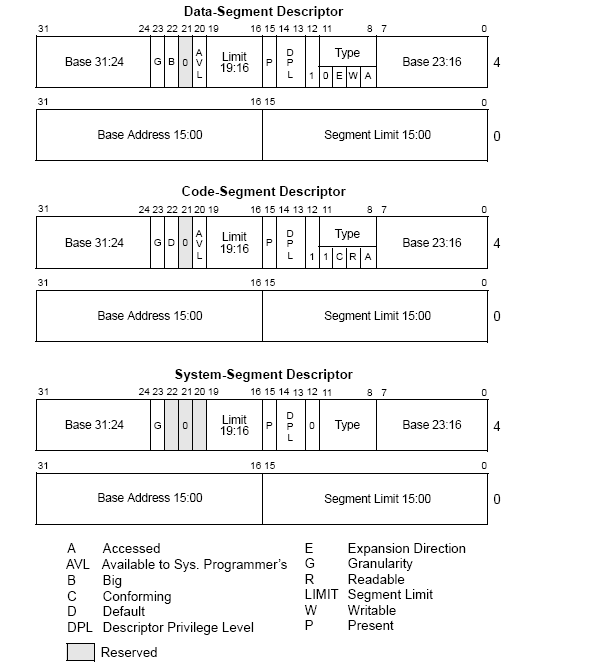
درزمانی که برنامه‌ای یک گزینشگر را در داخل ثبات قطعه بارگذاری می‌کند، پردازنده تنها فیلد پایه را در داخل ثبات قطعه بارگذاری نمی‌کند که علاوه بر آن پارامترهایی حفاظتی و دیگر پارامترها نیز در ثبات قطعه بارگذاری می‌شود. پردازنده اطلاعات لازم برای بررسی پارامترهای حفاظتی را از ثبات قطعه می‌خواند درنتیجه پالس ساعت اضافه‌ای مصرف نخواهد کرد.

### بررسی نوع

فیلد نوع در توصیفگرها دو کاربرد دارند

1. برای تمیز دادن ساختارهای متفاوت توصیفگرها
2. برای مشخص کردن کاربرد قطعات حافظه

معمولاً توصیفگرهای داده و توصیفگرهای کد توسط برنامه‌ها قابل‌دسترسی هستند ولیکن پردازنده توصیفگرهایی برای قطعات مخصوص دارند که فقط توسط سیستم‌عامل استفاده می‌شوند. جدول شماره 2 لیست انواع قطعات حافظه را در 80386 نشان می‌دهد. باید توجه کرد که همه توصیفگرها، قطعه‌های حافظه را توصیف نمی‌کنند. توصیفگرهای دروازه، برای هدف‌های دیگر طراحی‌شده‌اند که در بحث بعدی به آن اشاره می‌شود.



شکل شماره 9 - فیلدهای حفاظتی در انواع توصیفگرها

|  |  |
| --- | --- |
| reseverd | ۰ |
| Available 286 TSS | ۱ |
| LDT | ۲ |
| Busy 286 TSS | ۳ |
| Call gate | ۴ |
| Task gate | ۵ |
| 286 Intrruptgate | ۶ |
| 286 Trap gate | ۷ |
| reserverd | ۸ |
| Available 386 TSS | ۹ |
| reserved | A |
| Busy 386 TSS | B |
| 386 call gate | C |
| reserved | D |
| 386 Intrruptgate | E |
| 386 Trap gate | F |

جدول شماره 2 - لیست انواع قطعه حافظه در 80386

فیلد نوع در قطعات داده و کد شامل بیت‌های می‌شوند که تعریف‌های بیشتری در مورد اهداف قطعه می‌دهند.

* بیتی که قابلیت نوشتن در قطعه را مشخص می‌کند، در توصیفگرهای قطعات داده تعیین می‌کند که آیا دستوری می‌تواند در داخل این قطعه بنویسد
* بیتی که قابلیت نوشتن در قطعه را مشخص می‌کند، در توصیفگرهای قطعه کد مشخص می‌کند که آیا به دستورات اجازه داده‌شده از حافظه قطعه بخوانند. یک قطعه حافظه کد می‌تواند با دو روش خوانده شوند.
  1. از طریق ثبات CS
  2. به‌وسیله بارگذاری گزینشگرِ یک توصیفگر در داخل ثبات‌های قطعه داده (DS،ES،FS و یا GS)

از بررسی نوع می‌توان در کشف خطای برنامه‌ها استفاده کرد به این صورت دسترسی از راهی که در برنامه پیش‌بینی‌نشده باشد با شکست مواجه شود

پردازنده در دو فرصت اطلاعات فیلد نوع را بررسی می‌کند

1. درزمانی که گزینشگر یک توصیفگر در ثبات قطعه بارگذاری شود

ثبات‌های قطعه مشخص می‌توانند شامل تنها توصیفگرهای با نوع مشخصی باشند، برای مثال

* + ثبات CS تنها می‌تواند با گزینشگری که قطعه‌ای از نوع اجرایی را انتخاب می‌کند پر شود.
  + گزینشگرهای قطعه‌های اجرایی که بیت خواندنی بودن آن‌هاست نباشد نمی‌توانند در ثبات‌های قطعه داده بارگذاری شوند.
  + تنها گزینشگرهای قطعات باقابلیت خواندن می‌توانند در ثبات SS بارگذاری شوند.

1. زمانی که یک دستور به یک ثبات قطعه ارجاع دهد. قطعات به روش از قبل تعیین‌شده به‌وسیله دستورات مورداستفاده قرار می‌گیرند. برای مثال

* هیچ دستوری حق نوشتن در قطعات اجرایی را ندارد.
* هیچ دستوری حق نوشتن در قطعات داده، درزمانی که بیت نوشتنی آن ست نشده باشند را ندارد.
* هیچ دستوری حق خواندن از قطعات اجرایی، درزمانی که بیت خواندنی بودن آن ست نشده باشد را ندارد.

### بررسی محدوده

فیلد محدوده‌ای یک توصیفگر قطعه برای جلوگیری از آدرس‌دهی خارج از قطعه توسط پردازنده استفاده می‌شود. تفسیر پردازنده از فیلد محدوده وابسته است به وضعیت بیت G و علاوه بر آن در قطعات داده تفسیر پردازنده از فیلد محدوده وابسته به بیت E (بیت جهت گسترش[[22]](#footnote-22)) و همچنین بیت B (بیت بزرگ[[23]](#footnote-23)) می‌باشد.

در قطعات با خصوصت گسترش رو به پایین فیلد محدوده همان کاربرد را دارد ولی با یک تفسیر دیگر. در این حالت محدوده آدرس صحیح از مقدار فیلد محدوده شروع می‌شود و بسته به مقدار فیلد B تا 64k و یا تا 4GB گسترش میابد. یک قطعه حافظه از نوع گسترش رو به پایین وقتی حداکثر میزان را پیدا می‌کند که فیلد محدوده صفر گردد.

خصوصیت گسترش رو به پایین این امکان را فراهم می‌کند که اندازه پشته را بدون تغییر اشاره‌گر به پایه، افزایش دهیم.

فیلد محدوده در توصیفگری که مکانی از حافظه که جدول توصیفگرها در آن وجود دارد، می‌تواند مانع انتخاب مدخلی خارج از جدول شود. محدوده جدول توصیفگرها با محل آخرین توصیفگر معتبر در جدول مشخص می‌شود. هر توصیفگر هشت بایت طول دارد پس محدوده جدولی با طول N برابر است با N\*8-1

### سطح دسترسی

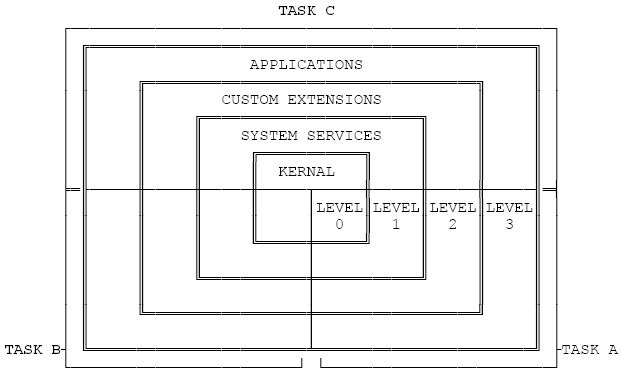
مفهوم دسترسی با نسبت دادن سه عدد از یک تا سه به اشیا تشخیصِ پردازنده، پیاده‌سازی شده و توسط پردازنده تشخیص داده می‌شود. این مقادیر به سطح دسترسی تعبیر می‌شوند. مقدار صفر بیشترین دسترسی و مقدار سه کمترین دسترسی را دارا می‌باشند. مواردی که تشخیص اجازه دسترسی در پردازنده به کمک آن‌ها صورت می‌گیرد شامل شامل موارد زیر می‌شود:

* توصیفگرها شامل فیلدی می‌باشند که سطح دسترسی توصیفگر[[24]](#footnote-24) گفته می‌شود.
* گزینشگرها شامل فیلدی می‌باشند که سطح دسترسی درخواست‌کننده[[25]](#footnote-25) نامیده می‌شود
* ثبات داخلی پردازنده که سطح دسترسی جاری[[26]](#footnote-26) را ثبت می‌کند. معمولاً سطح دسترسی جاری برابر با سطح دسترسی توصیفگرِ قطعه‌ای است که پردازنده در حال اجرای آن می‌باشد. سطح دسترسی تنها درزمانی تغییر می‌کند که کنترل به قطعه از حافظه، با سطح دسترسی توصیفگری، متفاوت تغییر می‌کند.

پردازشگر تعیین حق دسترسی روال جاری را به قطعه دیگر از حافظه به‌صورت خودکار انجام می‌دهد. برای تعیین حق دسترسی به قطعه‌ای دیگر پردازشگر سطح دسترسی جاری را با یک یا چندین دیگر از سطح دسترسی مقایسه می‌کند. ارزیابی در زمان بارگذاری گزینشگر یک توصیفگر به داخل ثبات قطعه انجام می‌شود. معیارهای ارزیابی دسترسی به داده با معیارهای دسترسی به کد تفاوت می‌کنند. درنتیجه دو روش دسترسی، هرکدام به‌صورت مجرا می‌باید بررسی شود.

شکل شماره 10 نشان می‌دهد که چگونه سطوح دسترسی به‌صورت حلقه‌های حفاظت تفسیر می‌شوند. حلقه مرکزی برای قطعاتی از حافظه می‌باشند که شامل برنامه‌های حیاتی و ضروری هستند. معمولاً هسته سیستم‌عامل از این جمله برنامه‌ها می‌باشد. حلقه‌های خارجی‌تر برای قطعاتی از حافظه است که شامل برنامه‌هایی با ضرورت کمتری برای سیستم به کار گرفته‌شده‌اند.

لزومی برای استفاده از هر چهار سطح دسترسی وجود ندارد. برنامه‌های موجود تنها برای استفاده از یک یا دو سطح دسترسی طراحی‌شده‌اند و بدون هیچ مشکلی دیگر سطوح دسترسی چشم‌پوشی می‌شوند. سیستم یک سطحی می‌باید از سطح صفر استفاده کند و سامانه‌های دوسطحی مبلید از سطح صفر و سطح یک استفاده می‌کنند.



شکل شماره 10 - حلقه‌های حفاظت

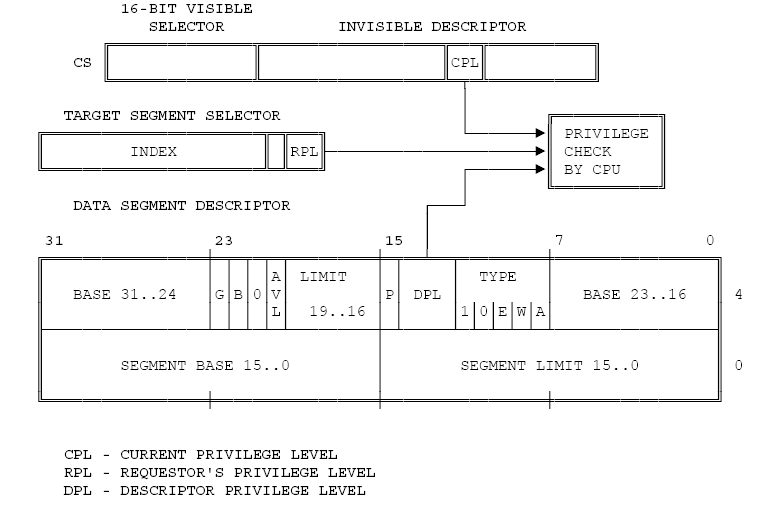
### محدودیت در دسترسی به داده

برنامه در حال اجرا در پردازنده برای استفاده از عملیات دسترسی به حافظه، گزینشگر قطعه داده موردنظر را می‌باید دریکی از ثبات‌های قطعه داده DS)، ES، FS، GS، (SS بارگذاری کند. پردازشگر ارزیابی دسترسی به حافظه داده را به‌صورت خودکار به به‌وسیله مقایسه سطوح دسترسی انجام می‌دهد. و این ارزیابی را در زمان بارگذاری گزینشگری برای توصیفگر قطعه مقصد در ثبات قطعه داده صورت می‌پذیرد. مطابق آنچه در شکل شماره 11 نشان داده‌شده، سه سطح دسترسی متفاوت در این نوع از بررسی اجازه دسترسی شرکت می‌کنند.

1. سطح دسترسی جاری
2. سطح دسترسی درخواست‌کننده گزینشگرِ استفاده‌شده برای مشخص کردن قطعه حافظه مقصد.
3. سطح دسترسی توصیفگرِ قطعه حافظه مقصد.

دستورات تنها درزمانی اجازه بارگذاری یک ثبات قطعه داده را پیدا می‌کنند که سطح دسترسی توصیفگر قطعه داده مقصد ازنظر عددی بزرگ‌تر یا برابر با بیشترین مقدار سطح دسترسی جاری و سطح دسترسی درخواستی باشد. به‌عبارت‌دیگر، یک‌روال در حال اجرا تنها درزمانی می‌توانند به داده دسترسی پیدا کند که سطح دسترسش برابر یا کمتر سطح دسترسی قطعه داده مقصد باشد.

ناحیه آدرس‌دهی بنا به مقدار سطح دسترسی جاری تغییر پیدا می‌کند. درزمانی که سطح دسترسی جاری برابر با صفر باشد. قطعات داده در تمامی سطوح دسترسی قابل‌دسترسی می‌باشند. و اگر سطح دسترسی جاری برابر با یک باشد تنها قطعات داده با سطح دسترسی یک تا سه قابل‌دسترسی می‌باشد. و در حالتی که سطح دسترسی جاری برابر با سه باشد تنها قطعاتی با سطح دسترسی سه قابل‌دسترسی می‌باشد.



شکل شماره 11- روند بررسی سطح دسترسی به قطعات داده

### محدودیت در انتقال کنترل

انتقال کنترل در پردازنده به‌وسیله دستورات JMP،CALL، RET،INT و IRET انجام می‌شود، به‌علاوه درزمانی که استثنا اتفاق میافتد و یا همچنین وقفه روی می‌دهد کنترل به روال استثنا و وقفه مربوطه منتقل می‌شود. بحث در مورد استثنا و وقفه خود نیاز به تعریف این مفاهیم دارد که خارج از دامنه مطالب این بخش می‌باشد. در این بخش به بررسی دستورات JMP،CALL و RET می‌پردازیم

شکل‌های نزدیک از دستورهای JMP،CALL و RET تنها قادر به انتقال کنترل در داخل قطعه کد جاری هستند. بنابراین تنها کافی است فیلد محدوده بررسی شود. پردازنده در این حالت از مقصد دستورات JMP،CALL، یا RET مطمئن می‌شود که احیاناً خارج محدوده مشخص‌شده توسط فیلد محدوده نباشد. فیلد محدوده در درون ثبات CS میانگیری شده است درنتیجه برای بررسی محدوده احتیاجی به پالس ساعت اضافه نیست.

حالت‌های دور از دستورات CALL،JMP به قطعه دیگر ارجاع می‌دهند. درنتیجه پردازنده سطح دسترسی را موردبررسی قرار می‌دهد. این دستورات به دو طریق، ارجاع به دیگر قطعات حافظه را میسر می‌کنند.

1. عملوند، قطعه حافظه کد دیگر را انتخاب کند.
2. عملوند، یک توصیفگر دروازه فراخوانی» را انتخاب کند.

این شکل دروازه‌ای از انتقال در ادامه همین فصل موردبررسی قرار می‌گیرد.

همان‌طور که شکل شماره 12 مشاهده می‌شود دو سطح دسترسی متفاوت در بررسی اجازه انتقال کنترل در حالت بدون استفاده از دروازه فراخوانی شرکت می‌کنند

1. سطح دسترسی جاری.
2. سطح دسترسی توصیف گر قطعه مقصد.

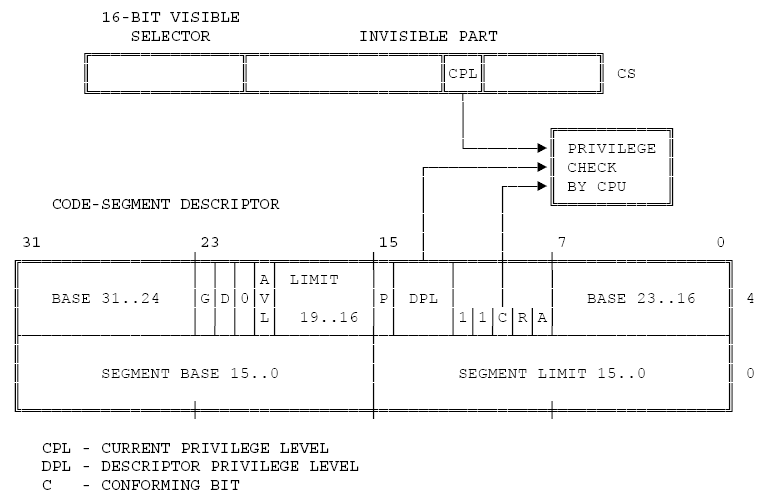
معمولاً سطح دسترسی جاری برابر با سطح دسترسی قطعه ایست که پردازنده در حال اجرای آن می‌باشد. اگرچه این امکان نیز وجود دارد سطح دسترسی جاری از سطح دسترسی توصیفگر قطعه جاری بزرگ‌تر باشد. برای به دست آوردن این حالت می‌باید بیت تطابق در توصیفگر قطعه جاری ست شده باشد. پردازنده رکوردی از سطح دسترسی جاری را در ثبات cs نگهداری می‌کند. و همان‌طور که پیش‌ازاین گفته شد مقدار نگهداری شده بعضاً می‌تواند با مقدار سطح دسترسی توصیفگر قطعه کد جاری متفاوت باشد.

پردازنده اجازه اجرای دستور JMP،CALL و پرش مستقیم به قطعه کد دیگر در صورتی می‌دهد که یکی از دو شرط تحقق یابد.

* سطح دسترسی توصیفگر قطعه مقصد برابر با سطح دسترسی جاری باشد.
* بیت تطابقی توصیفگر قطعه مقصد ست شده باشد و سطح دسترسی توصیفگر مقصد کوچک‌تر یا مساوی سطح دسترسی جاری باشد.

قطعه اجرایی که در توصیفگر آن بیت تطابقش ست شده باشد قطعه تطابقی گفته می‌شود. مکانیزم قطعه تطابقی برای اشتراک گذاشتن روال‌های است که امکان فراخوانی آن‌ها از سطح دسترسی‌های مختلف فراهم‌شده و در همان سطح دسترسی روال فراخوانندِ اجرا می‌شوند. برای مثال روالی را در نظر بگیرید که شامل کتابخانه‌های ریاضی و تعدادی راه‌انداز استثنا در آن قرارگرفته باشد. درزمانی که کنترل به قطعه تطابقی انتقال میابد سطح دسترسی جاری تغییر نمی‌کند.

خیلی از قطعات حافظه کد، تطابقی نیستند. قانون اولیه و بنیادی اجازه دسترسی این است که برای قطعات حافظه غیر تطابقی، انتقال کنترل بدون دروازه تنها می‌توانند به قطعه‌ای با سطح دسترسی مشابه صورت گیرد. به‌هرحال در بعضی از شرایط نیاز به انتقال کنترل به سطوح پایین‌تر دسترسی لازم می‌شود. بکار بردن دستور CALL همراه با توصیفگر دروازه فراخوانی» می‌تواند امکان انتقال کنترل به سطوح پایین‌تر را میسر سازد. دستور JMP هرگز نمی‌تواند کنترل را به قطعه غیر تطابقی با سطح دسترسی توصیفگری متفاوت از سطح دسترسی جاری انتقال دهد.



شکل شماره 12- بررسی اجازه دسترسی در انتقال کنترل

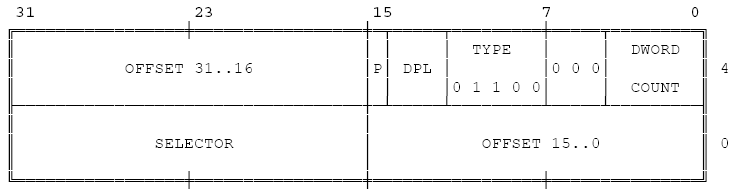
توصیفگرهای دروازه، نقطه شروع روال‌ها را محافظت می‌کنند

برای ایجاد حفاظت در حالت انتقال کنترل میان قطعات کد با سطح دسترسی متفاوت پردازنده از توصیفگر دروازه استفاده می‌کند. چهار نوع از توصیفگرهای دروازه به‌قرار زیر هستند.

* دروازه‌های فراخوانی
* دروازه‌های تله
* دروازه‌های وقفه
* دروازه‌های وظیفه

در این فصل تنها به دروازه‌ای فراخوانی می‌پردازیم. دروازه‌ای وظیفه در بحث چندوظیفگی مطرح می‌شوند. دروازه‌ای تله و وقفه نیز در بحث وقفه و استثنا موردبررسی قرار می‌گیرد.

ساختار یک دروازه فراخوانی درشکل شماره 13 نشان داده‌شده است. توصیفگر دروازه فراخوانی در هر دو نوع از جداول توصیفگر عمومی و یا محلی مستقر می‌گردند. ولیکن در جدول توصیفگر وقفه قرار نمی‌گیرند.



شکل شماره 13- ساختار توصیفگر دروازه فراخوانی

یک دروازه فراخوانی دو عملکرد اصلی دارد.

1. تعریف نقطه شروع روال‌ها.
2. مشخص کردن سطح دسترسی نقطه شروع روال‌ها.

توصیفگرهای دروازه فراخوانی توسط دستورهای CALL، JMP شبیه همان حالتی که در وجود توصیفگر قطعه کد استفاده می‌شد به کار گرفته می‌شوند. وقتی سخت‌افزار تشخیص داد که مقصد گزینشگر به یک توصیف گر دروازه ارجاع می‌دهد. عملکرد دستور بنا به محتوای دروازه فراخوانی مقصد تعیین می‌شود.

فیلدهای آفست و گزینشگر یک دروازه نقطه شروع روالی خاصی را مشخص می‌کنند. دروازه فراخوانی تضمین صحت کلیه انتقال‌ها به نقطه شروع‌هایی معتبر از روال‌های فراخوانده شده را می‌دهد؛ که در غیر این صورت امکان انتقال کنترل به وسط یک‌روال و یا در شرایطی بدتر به وسط یک دستور وجود دارد. عملوندهای اشاره‌گر به‌دوراز دستورهای انتقال کنترل به قطعه حافظه و فاصله از شروع آن اشاره نمی‌کنند. بلکه قسمت گزینشگر اشاره‌گر یک دروازه را انتخاب می‌کند؛ و فیلد آفست استفاده‌نشده باقی می‌ماند. همان‌طور که در شکل شماره 14 دیده می‌شود. چهار نوع سطح دسترسی متفاوت در بررسی اعتبار انتقال کنترل به‌وسیله دروازه فراخوانی صورت می‌پذیرد.

1. سطح دسترسی جاری
2. سطح دسترسی درخواستی گزینشگر استفاده‌شده برای مشخص کردن دروازه فراخوانی
3. سطح دسترسی درخواستی توصیفگر دروازه
4. سطح دسترسی توصیفگر قطعه حافظه مقصد

فیلد دسترسی توصیفگر دروازه تعیین می‌کند چه دسترسی می‌تواند از این دروازه استفاده کند. یک قطعه حافظه کد می‌تواند شامل چندین روال باشد که هرکدام برای یک سطح دسترسی خاص طراحی‌شده است. برای مثال سیستم‌عامل شامل تعدادی سرویس است که برای برنامه‌های کاربردی طراحی‌شده است درحالی‌که سرویس‌های دیگر نیز است که فقط برای استفاده خود سیستم‌عامل طراحی‌شده است.

دروازه‌های برای انتقال کنترل به سطوح پایین‌تر یا مشابه از دسترسی استفاده می‌شوند. که البته برای انتقال کنترل به سطوح مشابه احتیاج به دروازه نیست. تنها دستور CALL می‌تواند از دروازه برای انتقال کنترل به سطح پایین استفاده کند. دروازه‌ای که توسط دستور JMP استفاده می‌شود تنها می‌تواند انتقال را به سطوحی با سطح دسترسی مشابه انتقال دهد یا از قطعات حافظه تطابقی استفاده کند.

برای استفاده از دستور JMP می‌پایید این دو قانون زیر رعایت شود. و در غیر این صورت استفاده از این دستور با تولید یک استثنا مواجه می‌شود.

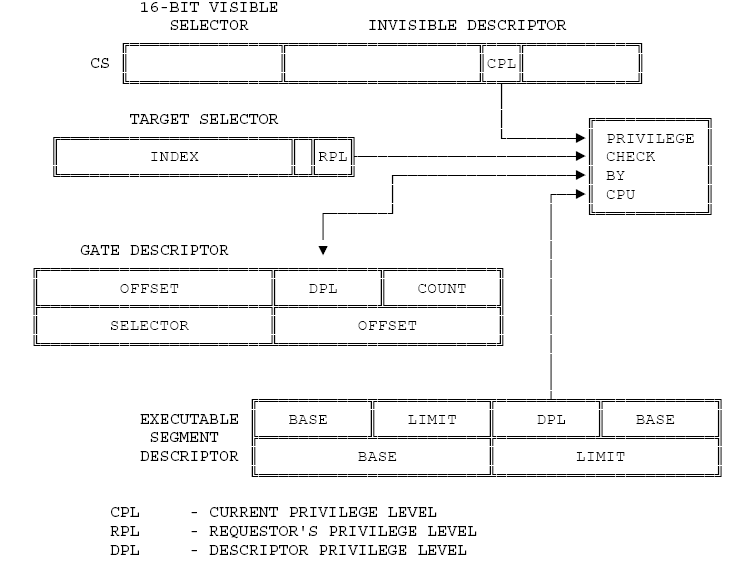
MAX (CPL،RPL) <= gate DPL

Target segment DPL = CPL

برای دستور CALL و یا دستور JMP به قطعه تطابقی می‌باید هر دو این قوانین تحقق یابند که در غیر این صورت یک استثنا عمومی ایجاد خواهد شد.

MAX (CPL،RPL) <= gate DPL

Target segment DPL <= CPL



شکل شماره 14- نحوه بررسی اجازه دسترسی در انتقال کنترل با استفاده از دروازه

### فراخوانی ‌های سریع

دستورات SYSENTER و SYSEXIT از زمان پردازنده‌های پنتیوم II به معماری IA\_32 اضافه گردید. این دستورات برای تأمین مکانیزمی سریع در فراخوانی سیستم‌عامل یا روال‌های اجرایی خاص دیگر بکار برده می‌شود.SYSENTER برای امکان دسترسی کدهای که در سطح دسترسی 3 در حال اجرا میباشند به سیستم‌عامل یا اجرای روال‌ها در سطح دسترسی صفر به کار گرفته می‌شود. دستور SYSEXIT در سطح دسترسی صفر در سیستم‌عامل یا روال اجرایی خاص دیگر برای بازگشت سریع به سطح دسترسی سه و کدهای اجرایی دیگر استفاده می‌شود. SYSENTER در تمامی سطوح دسترسی 3 تا 0 قابل‌اجرا میباشند. ولیکن SYSEXIT تنها می‌تواند در سطح دسترسی صفر اجرا شود.

دستورات SYSENTER،SYSEXIT دستورات هم‌خانواده‌ای هستند ولیکن مثل دستورات CALL/RET یک زوج را تشکیل نمی‌دهند و دلیلش این است که دستور SYSENTER هیچ‌گونه اطلاعاتی را در مورد وضعیت جاری حفظ نمی‌کند که در برگشت SYSEXIT از آن استفاده کند.

خط دستور مقصد و اشاره‌گر پشته در این دستورات به‌وسیله عملوندها مشخص نمی‌شود. بلکه به‌وسیله پارامترهایی که در s[[27]](#footnote-27)MSR و ثبات‌های عمومی واردشده‌اند، مشخص می‌گردند.

پارامترها در دستور SYSENTER از این منابع به دست می‌آیند.

* قطعه حافظه کد مقصد: از IA32\_SYSENTER\_CS خوانده می‌شود.
* دستور مقصد: از IA32\_SYSENTER\_EIP خوانده می‌شود.
* قطعه حافظه پشته: از جمع عدد 8 با مقدار IA32\_SYSENTER\_CS محاسبه می‌شود.
* اشاره‌گر پشته: از IA32\_SYSENTER\_ESP خوانده می‌شود.

پارامترها در دستور SYSEXIT از این منابع به دست می‌آید.

* جمع عدد 16 با مقدار IA32\_SYSENTER\_CS
* دستور مقصد: از ثبات EDX خوانده می‌شود.
* قطعه حافظه پشته: از جمع عدد 24 با مقدار موجود در IA32\_SYSENTER\_CS به‌دست‌آمده.
* اشاره‌گر پشته: از ثبات ECX خوانده می‌شود.

دستور SYSENTER و SYSEXIT به آن دلیل فراخوان سریع گفته می‌شود که به‌صورت پیش‌فرض با اجرا دستور SYSENTER کنترل به سطح صفر انتقال میابد و با اجرای دستور SYSEXIT به سطح سه انتقال میابد. درنتیجه احتیاج به بررسی اضافی در مورد سطح دسترسی وجود ندارد.

### دستورات ویژه

دستوراتی که بر ساختمان داده‌های سیستمی تأثیر می‌گذارند تنها می‌توانند درزمانی اجرا شوند که سطح دسترسی جاری برابر با صفر باشند. اگر پردازنده با این دستورات مواجه شود و سطح دسترسی جاری برابر با صفر نباشد. یک استثنا عمومی اتفاق می‌افتد. این دستورات شامل:

* LGDT - جدول توصیفگر عمومی بارگذاری ثبات
* LLDT- بارگذاری ثبات جدول توصیفگر محلی
* LTR- بارگذاری ثبات وظیفه
* LIDT- بارگذاری جدول توصیفگر وقفه
* MOV (control registers) بارگذاری و نگهداری ثبات‌های کنترلی
* LMSW- بارگذاری کلمه وضعیت ماشین
* CLTS پاک کردن پرچم تعویض وظیفه در ثبات CR0
* MOV (debug registers)- بارگذاری و نگهداری ثبات‌های اشکال‌زدا
* INVD- باطل کردن محتوای میانگیر
* WBINVD- باطل کردن محتوای میانگیر
* INVLPG-باطل کردن محتوای TLB
* HLT- متوقف کردن پردازنده
* RDMSR- خواندن ثبات‌های مدل – مخصوص
* WRMSR- نوشتن ثبات‌های مدل – مخصوص
* RDPMC- خواندن شمارنده دیدبان کارایی
* RDTSC- خواندن شمارنده مهر زمان

تعدادی از این دستورات ویژه اخیراً جز خانواده Intel 64،32- IA شده‌اند.

### دستورات حساس

دستوراتی که با ورودی و خروجی کار می‌کنند به اعمال محدودیت احتیاج دارند اما چون نیاز است که در سطح دسترسی غیر از صفر نیز اجرا شوند مکانیزم‌هایی خاصی در این دسته برای اعمال محدودیت در نظر گرفته‌شده که در بحث ورودی و خروجی جای می‌گیرند.

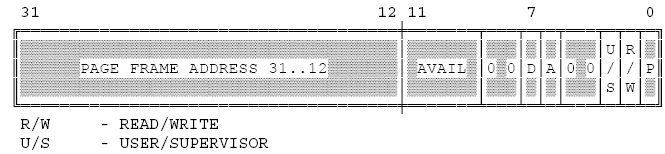
### حفاظت در سطح صفحه

# دو نوع از حفاظت در صفحه به کار گرفته می‌شود.

* محدودیت در ناحیه آدرس‌دهی
* بررسی نوع

مدخل‌های جدول صفحه پارامترهای حفاظتی را در خود نگهداری می‌کنند.

در شکل شماره 15 فیلدهای حفاظتی در مدخل جدول دیکشنری[[28]](#footnote-28) و مدخل جدول صفحه [[29]](#footnote-29)نشان داده‌شده است. این فیلدها کنترل دسترسی به صفحات را به عهده‌دارند.



شکل شماره 15- فیلدهای حفاظتی در صفحه

#### محدودیت در ناحیه آدرس‌دهی

مفهوم سطح دسترسی برای صفحات با دادن یکی از این دو سطح دسترسی پیاده‌سازی می‌گردد.

1. سطح سرپرستی[[30]](#footnote-30) (U/S=0)- برای سیستم‌عامل و دیگر برنامه‌های سیستمی و داده‌های مربوطه.
2. سطح کاربر (U/S=1)- برای برنامه‌های کاربردی ودادِ.

سطح جاری (U or S) به مقدار سطح دسترسی جاری (CPL) وابسته است. اگر CPL برابر با 0،1 یا 2 باشد پردازنده در حال اجرا در سطح سرپرستی هست و اگر CPL برابر با 3 باشد. پردازنده در سطح کاربری در حال اجرا هست.

وقتی پردازنده در حال اجرا در سطح سرپرستی است، همه صفحات قابل آدرس‌دهی میباشند؛ و اما اگر در حال اجرا در سطح کاربری باشد، تنها صفحات متعلق به سطح کاربری قابل آدرس‌دهی است.

#### بررسی نوع

در سطح آدرسی دهی به صفحه، دو نوع از صفحات تعریف‌شده‌اند.

1. دسترسی فقط خواندنی (R/W=0)
2. دسترسی خواندنی - نوشتنی (R/W=1)

درزمانی که پردازنده در سطح سرپرست در حال اجرا باشد. تمام صفحات هم خواندنی هستند و هم نوشتنی و زمانی که در سطح کاربر در حال اجرا است. تنها صفحاتی که سطح کاربر هستند و از نوع خواندنی – نوشتنی میباشند اجازه نوشته شدن دارند. در این حالت صفحاتی که به سطح سرپرستی تعلق دارند نه خواندنی هستند و نه نوشتنی.

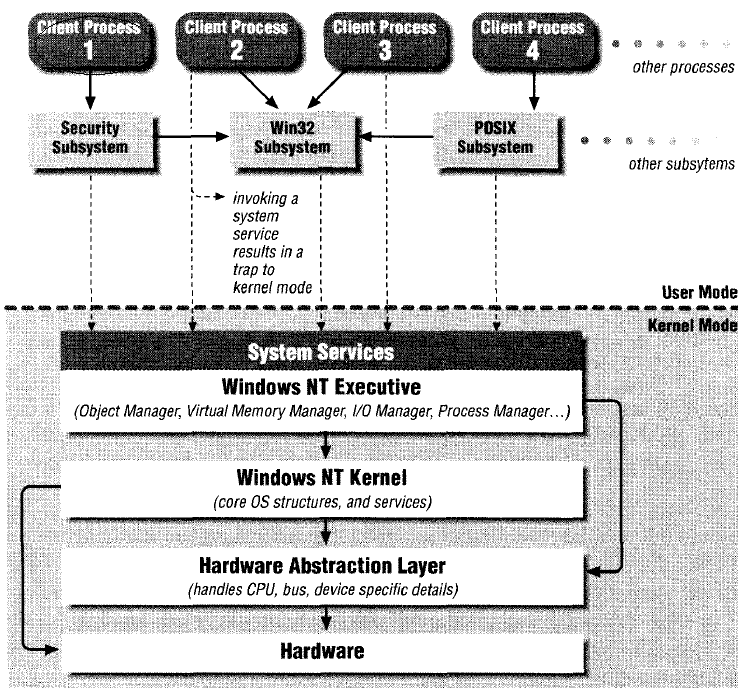
## معماری سیستم‌عامل[[31]](#footnote-31)

با بررسی مکانیزم مدیریت حافظه و همچنین حفاظت در پردازنده، معماری سیستم‌عامل به شکل مشخص‌تری فهمیده می‌شود.به علت آنکه سیستم‌عامل موردبررسی در این تحقیق سیستم‌عامل ویندوز هست در ادامه به بررسی اجمالی معماری و ساختار این سیستم‌عامل پرداخته می‌شود.

## معماری سیستم عامل ویندوز

یقیناً منطق مشتق شده از سیستم‌عامل MACH در طراحی سیستم‌عامل NT دیده می‌شود. این موارد شامل تلاش در کوچک کردن ناحیه KERNEL و پیاده‌سازی قسمتی از سیستم‌عامل به شکل مشتری – خدمت گذار همراه با استفاده از روش ارسال پیام برای انتقال اطلاعات مابین دو ماژول هست؛ و اما بیشتر از همه، طراحان سعی در استفاده از معماری لایه داشته‌اند.این سیستم‌عامل برای اجرا در هر دو حالت تک پردازنده‌ای و چندپردازنده‌ای متقارن طراحی‌شده است.

یکی دیگر از اهداف اصلی در طراحی ساختن این سیستم‌عامل این است که قابل‌حمل بر روی معماری‌های مختلف سخت‌افزار باشد. طراح برای رسیدن به این هدف از مدل شی گرا استفاده کرده است. در این حالت تعداد معدودی از اشیا به معماری سخت‌افزار وابسته میباشند و دیگر اشیا از مستقل از سخت‌افزار طراحی‌شده است. در تئوری بیشتر اشیا قابل‌حمل تنها با کامپایلی دوباره برای سخت‌افزارهای مختلف قابل‌استفاده هست. شکل شماره 16 چگونگی ساختار سیستم‌عامل NT را نشان می‌دهد. در این شکل مشاهده می‌شود که ویندوز به دو جز اصلی تقسیم‌شده است، حالت KERNEL و حالت کاربر.



شکل شماره 16- ساختار سیستم‌عامل ویندوز NT

دو نوع زیرسیستم در حالت کاربر وجود دارد. زیرسیستم‌های درونی که وظایف حیاتی سیستم‌عامل را انجام می‌دهد مانند زیرسیستم امنیتی و زیرسیستم‌های محیطی که رابط بین برنامه‌های کاربردی و هسته سیستم‌عامل هستند.

هر زیرسیستم (SUBSYSTEM) در فرآیند مربوط به خود اجرا می‌شود و حافظه جداگانه و محافظت‌شده از دیگر زیرسیستم‌ها دارد. هر فایل اجرایی تنها می‌تواند با یک زیرسیستم کار کند در زمان ایجاد یک فرایند نوع زیرسیستم فایل اجرایی فرآیند از قسمت هدر آن خوانده می‌شود و پس تعیین زیرسیستم به آن زیرسیستم ایجاد یک فرایند در آن زیرسیستم اطلاع داده می‌شود. نوع زیرسیستم را می‌توان در زمان لینک کردن با کلید /SUBSYSTEM مشخص کرد.

هر زیرسیستم واسط‌های برنامه‌نویسی کاربردی[[32]](#footnote-32) مخصوص به خود دارد. این واسط‌های برنامه‌نویسی توسط برنامه‌های کاربردی سطح کاربر که در آن زیرسیستم در حال اجرا است استفاده می‌شود. این واسط‌ها به‌صورت توابع مختلف که توسط فرایندها قابل‌دسترسی هستند پیاده‌سازی می‌گردند.

هر زیرسیستم به‌طور جداگانه با قسمت KERNEL سیستم‌عامل در ارتباط است. برای این کار با استفاده از جدول توزیع سرویس‌های سیستم[[33]](#footnote-33)، سرویس مدنظر خود در هسته سیستم‌عامل فراخوانی می‌کند. دسترسی به این جدول به دو طریق انجام می‌شود.

1. به‌وسیله وقفه به شماره 0x2E
2. با استفاده از دستور SYSENTER

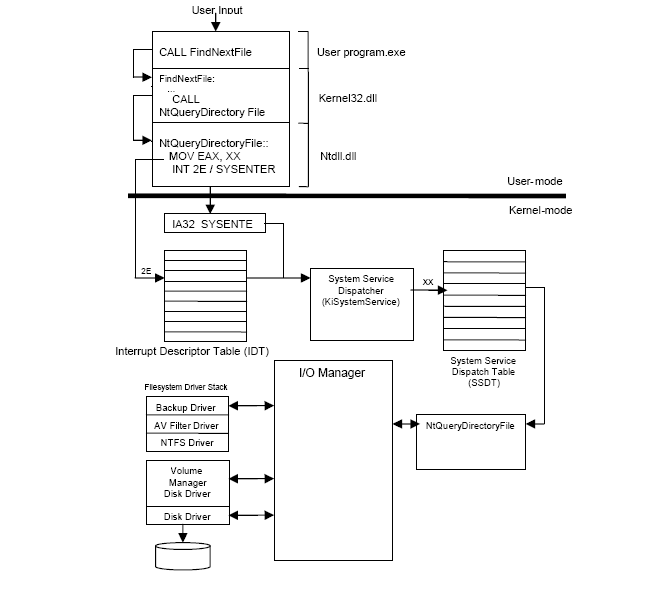
از ویندوز XP به بعد نوعاً از دستور SYSENTER استفاده می‌گردد. در NTهای قدیمی‌تر از وقفه شماره 0x2E استفاده می‌گردید. این دو مکانیزم کاملاً متفاوت هستند ولیکن به یک نتیجه ختم می‌شود. نتیجه هر دو روش به فراخوانی تابع KiSystemService در کرنل سیستم‌عامل منتهی می‌گردد. این تابع شماره سرویس موردنظر را از ثبات EAX خوانده و آن را در جدول توزیع سرویس‌های سیستم (SSDT) پیداکرده و فراخوانی می‌کند. همچنین تابع KiSystemService آرگومان‌های تابع فراخوانده شده را از پشته حالت کاربر به پشته حالت KERNEL انتقال می‌دهد. ثبات EDX به محل آرگومان‌ها اشاره می‌کند.

مشخصات جدول SSDT هیچ‌گاه توسط Microsoft منتشرنشده است؛ بنابراین برنامه‌ها برای کار مجبور به استفاده از زیرسیستم‌ها هستند و زیرسیستم‌ها با ایجاد لایه تجریدی بر روی هسته سیستم‌عامل برنامه‌ها را قادر می‌کنند که فارغ از پیاده‌سازی هسته به کار خود ادامه دهند. برای مثال ویندوزهای 9x و ME دارای ساختار هسته متفاوت با سیستم‌عامل‌های مبتنی بر NT هستند ولیکن چون هر دو از این دسته از زیرسیستم‌ها مشابه مانند WIN32 پشتیبانی می‌کنند تمامی برنامه‌ها مطابق با این زیرسیستم در هر یک از این دسته‌ها به‌طور کامل و بدون کم‌وکاست اجرا می‌شوند. زیرسیستم موجود در زمان بالا آمدن ویندوز در این مسیر رجیستری وجود دارد HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\SubSystems. از بین زیرسیستم‌ها موجود مانند posix، vdm زیرسیستم win32 محیط اجرایی بومی در ویندوز NT هست. ماکروسافت توسعه‌دهندگان برنامه‌های کاربردی را مصرانه به استفاده از APIهای این زیرسیستم برای فراخوانی سرویس‌های KERNEL تشویق می‌کند.

این زیرسیستم در مقایسه با دیگر زیرسیستم‌ها از سطح دسترسی بیشتری برخوردار است تا حدی که منحصراً مدیریت وسایل جانبی ارتباط با کاربر را به عهده دارد. مانیتور، کی‌برد، موس و همه دیگر وسایل ارتباطی توسط این زیرسیستم کنترل می‌شود. قوانین استفاده از واسط‌های گرافیکی نیز در این زیرسیستم وضع می‌شود. در حقیقت می‌توان گفت این تنها زیرسیستم موجود است که ماکروسافت از آن به‌طور کامل پشتیبانی می‌کند و دیگر زیرسیستم‌ها اگر هم وجود دارند بیشتر برای پر کردن لیست هستند تا اینکه عملیاتی باشند.

## روند یک فراخوانی سیستم

همان‌طور که گفته شد هر زیرسیستم APIهای خود را به واسط کتابخانه پیوند پویا[[34]](#footnote-34) در اختیار فرآیندها قرار می‌دهند. فرآیندها با فراخوانی توابع موجود در DLLها از امکانات موجود در سیستم‌عامل استفاده می‌کنند. شکل شماره 17 چگونگی فراخوانی یک API بانام FindNextFile نشان داده‌شده است.



شکل شماره 17- روند اجرای یک API

همان‌طور که در شکل مشاهده می‌شود برنامه کاربردی در حالت کاربر با فراخوانی یک تابع از KERNEL32.dll سرویس موردنظر خود را درخواست می‌کند. توابع فراخوانی شده در KERNEL32 نیز به‌نوبه خود تابع متناظرش را DLL.NTDLL فراخوانی کرده. در NTDLL توابع با دستور SYSCALL یا INT 0x2E تابع KiSystemService که یکی از توابع موجود Ntoskrnl.exe هست را فراخوانی می‌کند. این تابع همان‌طور که پیش‌ازاین گفته شد شماره سرویس درخواستی را از ثبات EAX خوانده (این مقدار پیش‌ازاین توسط تابع فراخوانده در NTDLL پرشده است) و سرویس (تابع) موردنظر را که آدرسش در جدول آمده فراخوانی می‌کند.

## راه‌انداز هسته

همان‌طور که در بخش مکانیزم حفاظت در پردازنده گفته شد، حافظه در حالت استفاده از مکانیزم صفحه‌بندی به دو ناحیه سرپرست و ناحیه کاربر تقسیم می‌شود. کد در حال اجرا در ناحیه کاربر نمی‌تواند دستورات ویژه را اجرا کند و همچنین نمی‌تواند به حافظه سرپرست دسترسی داشته باشد. و همه این محدودیت‌ها در ناحیه سرپرست از بین می‌رود.

در ویندوزهای مبتنی بر NT نوعاً نمی‌شود به‌عنوان یک برنامه‌نویس برنامه‌ای نوشت که در سطح دسترسی 0 و در ناحیه سرپرست اجرا شود مگر آنکه یک‌راه انداز حالت هسته[[35]](#footnote-35) نوشت.

راه‌اندازهای حالت هسته (Device Driver) به فایل‌های اجرایی گفته می‌شود که ساختاری به‌مانند کتابخانه‌های پیوند پویا(dll) دارند و توسط هسته در حافظه با دسترسی سیستمی بارگذاری شده تا بتوانند در کنار هسته سیستم به برنامه‌هایی سطح کاربری سرویس بدهند. پسوند فایل این ماژول‌ها عموماً sys است.که بیشتر در مسیر نصب ویندوز در پوشه system32/driver قرارگرفته‌اند.

راه‌اندازهای قطعات سخت‌افزاری مختلف به دلیل آنکه می‌باید مستقیم با سخت‌افزار در ارتباط باشند ازاین‌گونه ماژول‌ها میباشند و یا دیوارهای آتش و آنتی‌ویروس‌ها نیز به دلیل آنکه می‌باید کل ورودی و خروجی‌های سیستم را کنترل کنند، شامل قسمت‌هایی هستند می‌باید کد آن‌ها در هسته اجرا شوند درنتیجه از راه‌اندازهای حالت هسته استفاده می‌کنند.

برای بار گزاری یک ماژول راه‌انداز در سیستم‌عامل ویندوز می‌باید یک سرویس از نوع راه‌انداز ایجاد کرد. در زمان ایجاد یک سرویس تنظیماتی مانند نحوه اجرا و یا مسیر فایل اجرایی راه‌انداز مشخص می‌شود. برای ایجاد یک‌راه انداز سرویس در سیستم‌عامل ویندوز می‌باید دسترسی گروه مدیریت (administrator) را برای کاربر در حال اجرا وجود داشته باشد. این سیاست از مهم‌ترین سیاست‌های امنیتی است که برای جلوگیری از دسترسی غیرمجاز به هسته سیستم‌عامل استفاده می‌شود.دسترسی غیرمجاز به هسته باعث می‌شود کل سیستم دچار تحدید امنیتی جدی شود. درنتیجه ممیزی‌های امنیتی بیشتری ازآنچه توضیح داده شد در سیستم‌عامل ایجادشده است.

دسته‌ای از بدافزارها وجود دارند که سعی در دور زدن سیاست‌های امنیتی موجود برای دسترسی به هسته عامل دارند. در ادامه به معرفی آن‌ها می‌پردازیم.

## معرفی مکانیزم ماشین مجازی در پردازنده‌ها

مجازی‌سازی برپایِ سخت‌افزار اولین بار در سال 1974 در بزرگ‌رایانه[[36]](#footnote-36) IBM system/370 به کار گرفته شد. بر روی این کامپیوتر اولین سیستم‌عامل مجازی‌سازی با عنوان vm/370 نصب می‌شد. قابلیت مجازی‌سازی بر روی پردازنده‌های سری POWER توسط IBM ایجاد شد و در بزرگ‌رایانه‌های این شرکت امکان به‌کارگیری داشته و دارد.

Gerald J. Popek و [14]Robert P. Goldberg در مقاله‌ای با عنوان الزاماتی برای شبیه‌سازی در معماری نسل سوم[[37]](#footnote-37) که در سال 1974 به چاپ رسید. پس از معرفی ویژگی‌های اساسی یک ماشین مجازی، قانونی را بیان می‌کند که باید در معماری سیستم وجود داشته باشد تا بتوان در آن سیستم مجازی‌سازی بر اساس سخت‌افزار صورت گیرد. ویژگی‌هایی که در این مقاله برای یک ماشین مجازی معرفی‌شده است به شرح زیر است.

هم ارزی: اجرای یک برنامه در درون یک ماشین مجازی باید اساس رفتار یکسان بازمانی که در یک ماشین واقعی اجرا می‌شود داشته باشد. البته برای این حالت استثناهایی را نیز در نظر می‌گیرد؛ مانند رفتارهایی که وابسته به‌سرعت اجرا باشد و یا میزان منابع در دسترس برنامه. دلیل این دو استثنای هم کاملاً واضح است. به علت اینکه استفاده از پردازنده و منابع در دسترس در مجازی‌سازی به اشتراک گذاشته می‌شود، سرعت اجرا و میزان منابع در ماشین مجازی با سیستم واقعی برابری نمی‌کند.

کنترل منابع: یک ماشین مجازی باید کنترل کاملی بر روی منابع مجازی‌سازی شده داشته باشد.

کارایی: بخش عمده‌ای از دستورات ماشین می‌باید بدون دخالت ماشین مجازی اجرا شوند.

قضیه‌ای رو هم در مقاله خود مطرح و اثبات کردند به شرح زیر است:

برای هر کامپیوتر نسل 3، ماشین مجازی کارای می‌تواند درست شود اگر مجموع دستورات حساس برای آن کامپیوتر زیر مجموع دستورات ممتاز آن قرار گیرد. مجموع دستورات ممتاز به مجموع دستورات گفته می‌شود که می‌توان برای آن تله ایجاد کرد. به این شکل که با اجرا آن دستور در سیستم یک‌روال فراخوانی می‌شود. مجموعه دستورات حساس نیز به مجموع دستوراتی گفته می‌شود که رفتار آن‌ها وابسته به وضعیت منابع سیستم است. به بیانی ساده‌تر مجموعه دستوراتی که وابسته تنظیمات سیستم هستند و یا تنظیمات سیستم را تغییر می‌دهند می‌باید تله داشته باشند. این قضیه یک روش ساده و کارا برای شبیه‌سازی معرفی می‌کند، روش تله و تقلید[[38]](#footnote-38)*.*

با افزایش ظرفیت‌های پردازشی در سرورهای x86 استفاده از مجازی‌سازی بر روی آن‌ها نیز توجیه پیدا می‌کرد. با مجازی‌سازی امکان به اشتراک‌گذاری منابع سیستم بین چند سیستم مجازی به وجود می‌آید و درنتیجه بهره‌وری سیستم نیز افزایش پیدا می‌کند. ولیکن به علت اینکه مجموعه دستورات اولیه x86 شروط گفته‌شده در قضیه بیان‌شده در کار Gerald J. Popek و Robert P. Goldberg نداشته‌اند امکان ایجاد ماشین مجازی سخت‌افزاری بر روی آن‌ها وجود نداشت. البته برنامه‌هایی مبتنی بر مجازی‌سازی نرم افرازی برای این معماری ایجاد شد که از کارایی لازم برخوردار نبودند. به‌عنوان‌مثال می‌توان از برنامه vmware نام برد که در سال 1999 نسخه ایستگاه کاری VMware Workstation و در سال 2001 نسخه سروری VMware GSX Server خود را ارائه کرد که هر دو نسخه مبتنی بر روش مجازی‌سازی نرم‌افزاری عمل می‌درند.

در سال 2005 و 2006 میلادی دو شرکت بزرگ تولیدکننده پردازنده‌های x86، AMD و Intel به شکل جداگانه افزونه جدید به معماری x86 اضافه کردند که امکان مجازی‌سازی سخت‌افزاری را بر روی سیستم ایجاد می‌کرد. شرکت Intel نام فنّاوری خود را Intel-vt گذاشت و نام AMD-V را شرکت AMD برای این فنّاوری انتخاب کرد.

از آن زمان بود که برنامه‌های ناظر ماشین مجازی برای این معماری توسعه داده شد. از معروف‌ترین این برنامه‌ها Linux Kvm و Microsoft Hyper-v، XenCenter، XEN3.0 و VMware می‌توان نام برد.

از بین دو فنّاوری معرفی‌شده برای معماری x86، در این تحقیق تمرکز بر روی فنّاوری Intel-vt هست. در ادامه به معرفی اجمالی این فنّاوری پرداخته می‌شود.

### معماری ماشین مجازی

افزونه ماشین مجازی اینتل (VMX[[39]](#footnote-39)) دو کلاس نرم‌افزاری را پشتیبانی می‌کند.

1. نرم‌افزار ناظر ماشین مجازی[[40]](#footnote-40): VMM به‌عنوان میزبان در سیستم عمل می‌کند و دسترسی کامل به پردازنده و دیگر سخت‌افزارها دارد. VMM ماشین میهمان (در ادامه توضیح داده می‌شود) را در سیستم ایجاد کرده و یک پردازنده انتزاعی برای آن تعریف می‌کند و اجازه می‌دهد که ماشین مهمان بر روی آن اجرا شود. VMM می‌تواند که کنترل خاص بر روی پردازنده، حافظه فیزیکی، وقفه‌ها و ورودی و خروجی، برای خود تعریف و ایجاد کند.
2. نرم‌افزار مهمان: مجموعه‌ای از نرم‌افزار که در محیط ایجادشده توسط VMM کار می‌کنند.

همچنین VMX دو نوع مد عمل کردی متفاوت دارند. مد عمل کردی ریشه و مد عمل کردی غیر ریشه. VMM در مد عملکردی ریشه اجرا می‌شود و ماشین مهمان در مد عمل کری غیر ریشه. به انتقال VMX از مد عملکردی غیر ریشه به مد عملکردی ریشه خروج VM گفته می‌شود و به انتقال از مد عملکردی ریشه به غیر ریشه، انتقال به مدخل VM گفته می‌شود.

رفتار پردازنده در مد عمل کردی ریشه بسیار شبیه زمانی است که مد VMX فعال نشده باشد. تفاوت اصلی بین این دو حالت درمجموع دستورات است که مد ریشه VMX در دسترس قرار می‌گیرد و همچنین محدودهایی مقداردهی بعضی از ثبات کنترلی سیستم در زمان مد ریشه فعال باشد.

رفتار پردازنده درزمانی مد عملکردی ریشه فعال نباشد به‌منظور امکان ایجاد مجازی‌سازی، محدود و تغییر پیدا می‌کند. در این حالت بعضی از دستوارت (مانند دستور VMCALL) و رویدادها باعث خروج VM می‌شود. به دلیل آنکه خروج VM جایگزین رفتار عادی پردازنده می‌شود. عملکرد نرم‌افزار در مد عملکردی غیر ریشه محدودشده است. این محدود اجازه می‌دهد کنترل منابع پردازنده را حفظ کند.

هیچ علامت قابل دیدی برای نرم‌افزار که مشخص کند پردازنده در مد عملکردی غیر ریشه فعال است وجود ندارد. این قابلیت به VMM این اجازه را می‌دهد که برنامه میهمان نتواند تعیین کند در ماشین مجازی در حال کار است.

## چرخه فعالیت برنامه VMM

چرخه فعالیت یک ماشین مجازی شامل چهار بخش می‌شود:

1. برنامه با اجرای دستور VMXON وارد مد عملکردی VMX می‌شود.
2. برنامه VMM می‌تواند با به‌کارگیری دو دستور VMLUNCH و VMRESUME یک سیستم مهمان را به‌صورت مجازی اجرا کند
3. VMM با تعریف رویداد، کنترل را در زمان‌هایی که آن رویدادها در ماشین مجازی روی می‌دهد به دست آورد.
4. درصورتی‌که برنامه بخواهد از مد عمل کردی VMX خارج شود از دستور VMXOFF استفاده می‌کند

## معرفی اجمالی ساختار VMCS

عملیات غیر ریشه در VMX و انتقال‌ها در آن به‌وسیله ساختمان داده مخصوصی کنترل می‌شود. نام آن، ساختمان داده کنترل ماشین مجازی (virtual-machine control structure) است که به‌اختصار به آن VMCS گفته می‌شود.

در X86 دو دستور VMPTRST و VMPTRLD برای بارگزاری و بازنشانی این ساختمان داده در حافظه استفاده می‌گردد؛ و دو دستور VMREAD و VMERITE برای عملیات خواندن و نوشتن برای روی این ساختمان داده مورداستفاده قرار می‌گیرد. در ادامه به شرح مختصری از ساختار این ساختمان داده پرداخته می‌شود.

## ساختمان داده VMCS

داده‌های موجود در این ساختمان داده را می‌توان در شش گروه تقسیم‌بندی کرد. این شش گروه به شرح زیر است:

ناحیه وضعیت مهمان: وضعیت پردازنده در زمان عملیات خروج vm از این ناحیه خوانده و برگزاری می‌گردد و در زمان عملیات انتقال به مدخل vm در این ناحیه ذخیره می‌گردد.

اطلاعاتی که از وضعیت پردازنده در این ناحیه ذخیره می‌گردد شامل موارد زیر است.

ثبات‌های کنترلی شامل: CR3 ، CR0،CR4

ثبات‌های اشکال‌زدایی: ِ DR7

ثبات‌های RIP،RSP،RFLAGS و همچنین گزینشگرهای توصیفگر CS،SS،DS،ES،FS،GS،LDTR،TR

تعدادی از ثبات‌های خاص مدل شامل IA32\_DEBUGCTL، IA32\_SYSENTER\_CS ...

...

همه فیلدها در هر بار انتقال به مدخل VM خوانده‌شده و در ثبات‌های متناظر نوشته می‌شود و همچنین در هر بار خروج vm در مقادیر متناظر ذخیره می‌گردد.

ناحیه وضعیت میزبان: وضعیت پردازنده در زمان عملیات خروج vm از این داده‌های خوانده می‌شود.. ثبات‌ها و دیگر مقادیری که در این ناحیه ذخیره می‌شود، تقریباً همان‌هایی هستند که در ناحیه وضعیت مهمان نگهداری می‌شود.

فیلدهای کنترل اجرای vm: این فیلدها رفتار پردازنده را در زمان اجرای عملیات غیر ریشه vm کنترل می‌کنند. این فیلدها قسمتی از موارد خروج از vm را تعیین می‌کنند. مهم‌ترین این فیلدها شامل موارد زیر است:

* تعریف رویدادهای خروج غیرمتقارن مانند تعریف خروج برای زمانی که وقفه‌ای روی دهد.
* تعریف رویدادهای خروج متقارن، مانند تعریف رویداد خروج براثر اجرای دستوری که تغییر در مقدار ثبات‌های اشکال‌زدایی ایجاد کند.
* تعریف خروج براثر روی دادن استثنا. می‌توان برای هر یک از استثناهای موجود در سیستم رویداد خروج تعیین کرد.
* تعریف رویداد خروج براثر اجرای دستورات ورودی و خروجی.
* تعریف ماسک و مقدار سایه برای ثبات‌های کنترلی cr0 و cr4. اگر مقدار نوشته‌شده بر روی این ثبات‌ها با مقدار تعیین‌شده برابر نباشد ایجاد رویداد خروج می‌کند؛ و همچنین با استفاده از این فیلدها می‌توان تعیین کرد که مقدار خوانده‌شده توسط ماشین مجازی با مقدار واقعی آن در پردازنده مجازی متفاوت باشد.
* کنترل کردن مقدار ثبات cr3: می‌توان تعدادی مقدار را برای cr3 تعیین کرد و درصورتی‌که این ثبات مقدار متفاوت از این مقادیر را بگیرد رویداد خروج ایجاد می‌شود.

فیلدهای کنترل عملیات خروج vm: این فیلدها عملیات خروج vm را کنترل می‌کنند. رویدادهایی که در زمان خروج vm اتفاق می‌افتند در این فیلدها قابل‌تعریف هستند. به مثال می‌توان تعیین کرد که مقدار ثبات‌های اشکال‌زدایی در زمان خروج vm ذخیره شوند.

فیلدهای کنترل عملیات انتقال به مدخل vm: این فیلدها عملیات انتقال به مدخل vm را کنترل می‌کنند. رویدادهایی که در زمان انتقال به مدخل vm اتفاق می‌افتند در این فیلدها قابل‌تعریف هستند. به مثال می‌توان تعیین کرد که مقدار ثبات‌های اشکال‌زدایی در زمان انتقال به مدخل vm برگزاری شوند.

فیلدهای نشان‌دهنده اطلاعات خروج vm: این فیلدها اطلاعاتی مربوط به علت خروج vm را خود نگهداری می‌کند. این اطلاعات شامل اطلاعاتی در مورد دستور و یا وقفه‌ای که باعث ایجاد خروج vm شده است، می‌شود.

فیلدهای کنترل اجرای vm، کنترل عملیات خروج VM و کنترل عملیات انتقال به مدخل vm را درمجموع فیلدهای کنترلی گویند.

# روت‌کیت [[41]](#footnote-41)

نام rootkit از دو جز root که اشاره به کاربری با دسترسی حداکثر در سامانه‌های unix دارد و همچنینkit به معنای برنامه کوچک در نقش ابزار دم‌دستی از آن استفاده می‌شود، تشکیل‌شده است. تفاوت اصلی روت‌کیت‌ها با دیگر بدافزارها در این است که پنهان‌سازی در آن هدف اصلی هست. روت‌کیت‌ها به دودسته کلی تقسیم‌بندی می‌شوند. روت‌کیت‌های سطح کاربری و روت‌کیت‌های سطح هسته. در این تحقیق روت‌کیت‌های سطح هسته موردتوجه قرار می‌گیرند. روت‌کیت‌های سطح کاربری با استفاده از واسط‌های ارائه‌شده توسط سیستم‌عامل در سطح کاربری، بعضی از روش‌های مخفی سازی ممکن را اجرا می‌کنند. بررسی آن‌ها و روش‌های جلوگیری از آن‌ها نیاز به تحقیق جداگانه دارد.

در ادامه متن، به‌جای استفاده از روت‌کیت‌های سطح هسته به‌اختصار همان روت‌کیت مورداستفاده قرار می‌گیرد.

روت‌کیت‌ها در بیشتر مواقع به‌عنوان جزی از یک بسته بدافزاری در سیستم هدف مورداستفاده قرار می‌گیرند. بدافزارها از این ماژول‌ها استفاده می‌کنند تا خود را در سیستم هدف به شکل کامل مخفی سازی کنند.

روت‌کیت‌ها انواع مختلف دارند که چندی از آن‌ها در ادامه توضیح داده می‌شوند.

روت‌کیت‌های فری[[42]](#footnote-42)م‌ور- این دسته از روت‌کیت‌ها از سخت‌افزارهایی که فریم‌ور دارند استفاده می‌کنند تا کد خود را در درون آن‌ها قرار دهند. به علت آنکه دسترسی به فریم‌ورهای سخت‌افزاری نسبتاً مشکل است و اغلب معتبر بودن آن‌ها بررسی نمی‌شود، این نوع از روت‌کیت‌ها امکان پنهان‌سازی بسیار زیاد و امکان کشف بسیار کمی دارند. البته توسعه و ساخت این دسته از روت‌کیت‌ها با پیچیدگی بسیار همراه است. وابستگی به نوع خاصی از سخت‌افزار نیز از محدودیت‌های آن‌ها میباشند.

روت‌کیت ماشین مجازی- این روت‌کیت‌ها از امکانات سخت‌افزاری موجود در پردازنده‌ها استفاده می‌کند تا بتوانند سیستم‌عامل را به‌صورت ماشین مجازی اجرا کنند و خود به‌عنوان ماشین اصلی کنترل ورودی و خروجی سیستم‌عامل را به دست بگیرند. به علت آنکه این دسته از روت‌کیت‌ها کاملاً از دید سیستم‌عامل مخفی هستند و از درون ماشین مجازی به حافظه آن‌ها هیچ دسترسی نیست درنتیجه این دسته از روت‌کیت‌ها از قابلیت مخفی سازی بسیار زیادی بهره می‌برند و کشف آن‌ها را نیز بسیار مشکل است. ازنظر وسعت دسترسی نیز این دسته از روت‌کیت‌ها تقریباً می‌توانند کل سیستم را در اختیار بگیرند و دسترسی سیستم‌عامل را نیز محدود کنند.

روت‌کیت‌های راه‌اندازها[[43]](#footnote-43) – این دسته از روت‌کیت‌ها با آلوده کردن مسیر راه‌اندازی سیستم، قبل از سیستم‌عامل اجرا می‌شوند و در ادامه در هنگام برگزاری سیستم‌عامل به داخل آن نفوذ می‌کنند.

روت‌کیت‌های از نوع ماژول هسته- روت‌کیت‌های از نوع ماژول هسته اصلی‌ترین نوع روت‌کیت‌ها میباشند. بعد از برگزاری این ماژول در فضای حافظه هسته سیستم این ماژول‌ها قادر هستند هر کاری در سیستم انجام دهند؛ اما شایع‌ترین رفتارهای این بدافزارها شامل: مخفی سازی فرایندهای بدافزارها. مخفی سازی فایل‌های بدافزارها. استراق از ترافیک شبکه. مخفی سازی اتصالات شبکه برای بدافزارها و ...

شناسایی و حذف روت‌کیت‌ها در سیستم‌عامل عمل بسیار پیچیده و زمان بری هست. در بعضی از مواقع نیاز به مداخله سخت‌افزار نیز هست مانند زمانی که سیستم به یک روت‌کیت فریم‌ور آلوده‌شده باشد.

## روال محافظتی در برابر روت‌کیت‌ها

روش‌های محافظت از هسته سیستم‌عامل به دودسته کلی تقسیم می‌شود.

1. روش‌های جلوگیری از نصب و برگزاری ماژول غیر معتبر در هسته سیستم‌عامل
2. روش‌های شناسایی رفتار و تغییرات مخرب در فضای هسته سیستم‌عامل

روش‌های جلوگیری از نصب و بارگزاری ماژول‌های غیر معتبر در هسته دو راه‌کار کلی دارد. یک آنکه تنها دسترس‌های خاصی بتواند ماژول هسته در سیستم معرفی و نصب کنند؛ و دیگر آنکه تنها فایل‌هایی با امضای دیجیتال معتبر بتواند در سیستم نصب شود.

### روش‌های جلوگیری از نصب و برگزاری ماژول غیر معتبر در هسته سیستم‌عامل

روش بررسی دسترسی برای نصب سرویس از زمان ویندوز 2000 مطرح بوده و در ابتدا بیشتر در حفظ یکپارچگی سیستم و نظم سیستم به کار گرفته‌شده است و در ادامه بااهمیت پیدا کردن موضوع امنیت به‌عنوان یک مکانیزم امنیتی نیز موردنظر بوده است. درحالی‌که حاضر تنها کاربر با دسترسی مدیر می‌تواند سرویس نصب کند. به علت آنکه ماژول‌های هسته به‌عنوان یک سرویس به سیستم‌عامل معرفی می‌شوند درنتیجه امکان نصب و اجرای ماژول‌های هسته برای غیر مدیر ممکن نیست.

روش‌هایی اعتبار سنجی بر اساس امضای دیجیتال برای فایل‌های ماژول‌های هسته در سیستم عامل ویندوز مورداستفاده قرار گفته است [10]. در این روش با استفاده از روش‌های درهم سازی مانند sha1 یک‌رشته چند بایتی متناظر با فایل ایجاد می‌شود که به همراه اندازه فایل یک مقدار یکتا که مشخصه آن فایل است ایجاد می‌گردد. در ادامه این مقدار با یک کلید خصوصی که توسط شرکت‌هایی اعتبار سنجی ایجادشده است و برای شرکت مایکروسافت نیز هویت آن‌ها احرازشده‌اند کد می‌شود. کلید خصوصی تنها به شرکت‌های معتبر و یا افراد با اطلاعات هویتی کامل واگذار می‌گردد. مقدار رمز شده در درون فایل ماژول قرار می‌گیرد. در زمان برگزاری ماژول در حافظه این مقدار توسط کلید عمومی موجود در سیستم‌عامل بازشده و مقدار یکتای ذخیره‌شده در فایل به دست می‌آید. درهم‌سازی بر روی مقادیر فایل انجام می‌گیرد و با مقدار به دست آماده از رمزگشایی بررسی می‌گردد در صورت تساوی دو مقدار نتیجه می‌شود که فایل تغییر نکرده و معتبر است.

### روش‌های شناسایی رفتار و تغییرات مخرب در فضای هسته سیستم‌عامل

روال‌هایی در سیستم‌عامل نسخه xp شصت چهار بیتی به بعد ایجاد شد که با بررسی یکپارچگی هسته سیستم‌عامل به شکل دوره‌ای و در بازه‌های زمانی غیرمنظم مشخص می‌کند که آیا هسته دچار تغییراتی شده است که باعث ضعف امنیتی در سیستم شده باشد یا خیر. نام این مکانیزم در سیستم‌عامل ویندوز Patch Guad است [11].

در دوره‌های زمانی ثابت این مکانیزم شروع کارکرده و نقاطی مشخص از تصویر هسته در حافظه را موردبررسی قرار می‌دهد در صورتی این برنامه متوجه تغییر در تصویر هسته در حافظه شود که این تغییر باعث ایجاد یک اشکال امنیتی در سیستم شود سیستم را با نمایش صفحه آبی برگزاری مجدد می‌کند.

## روش‌های مورداستفاده روت‌کیت‌ها در دور زدن مکانیزم‌های امنیتی

### افزایش دسترسی [[44]](#footnote-44)

در مراحل ابتدایی نفوذ اغلب دسترسی‌ها به سیستم به‌حساب کاربری غیر مدیر صورت می‌گیرد. همان‌طور هم که قبلاً بیان شد برای نصب ماژول‌های هسته می‌باید دسترسی مدیر برای حساب کاربری نصب کننده وجود داشته باشد. درنتیجه اغلب به‌صرف نفوذ و دسترسی غیرمجاز به سیستم نمی‌توان روت‌کیت نصب کرد. در این مواقع نیاز به افزایش دسترسی جاری به دسترسی مدیر به شکل غیرمجاز می‌باشد.

افزایش دسترسی عملاً با بهره‌برداری و سو استفاده (exploiting) از یک اشکال در سیستم‌عامل صورت می‌گیرد. این اشکال می‌تواند در طراحی و یا پیاده‌سازی صورت گرفته باشد. برای مثال در نسخه‌های اولیه سیستم‌عامل ویندوز محافظ صفحه‌نمایش با دسترسی مدیر اجرا می‌شد. با جایگزین کردن فایل محافظ صفحه‌نمایش اصلی با فایل آلوده، با اجراشدن محافظ صفحه‌نمایش فایل آلوده با دسترسی مدیر اجرا می‌شد. این اشکال در طراحی به علت در نظر نگرفتن دسترسی کاربر فعال در اجرای محافظ صفحه‌نمایش می‌تواند مورد بهره‌برداری توسط نفوذکننده قرار گیرد و افزایش دسترسی از یک کاربر غیر مدیر به یک کاربر مدیر را صورت دهد.

برای بررسی یک مثال عملی می‌توان به اشکال گزارش‌شده با شناسه CVE-2014-4113 و مقاله Ronnie Johndas [12]که برای این اشکال تهیه‌کرده رجوع کرد.

پس از افزایش دسترسی می‌توان ماژول هسته را نصب کرد ولیکن به علت وجود مکانیزم اعتبار سنجی، تنها ماژول‌های معتبر امکان نصب و راه‌اندازی را دارند درنتیجه برای نصب روت‌کیت باید روال‌های اعتبار سنجی ماژول‌ها را از کار انداخت.

### غیرفعال سازی روش‌های اعتبار سنجی ماژول‌های هسته

روال‌های اعتبار سنجی ماژول هسته در خود هسته و در ناحیه حافظه با دسترسی ممتاز (سطح 0) کار می‌کنند. دو دلیل را می‌توان برای این حالت ذکر کرد.

* 1. امکان اعتبار سنجی ماژول‌های هسته درزمانی که سیستم در حال برگزاری اولیه خود است و هنوز فضای برنامه‌های کاربردی ایجاد نشده است.
  2. جلوگیری از دسترسی برنامه‌های کاربردی به حافظه روال‌های اعتبار سنجی.

برای غیرفعال سازی روال‌های اعتبار سنجی دسترسی به حافظه ناحیه هسته اجتناب‌ناپذیر است؛ اما خود روال‌ها اعتبار سنجی امکان اجرای ماژول‌های غیر معتبر را نمی‌دهند درنتیجه دسترسی به هسته به‌وسیله ماژول غیرممکن است. در این حالت می‌باید روش دیگری غیر از بارگزاری ماژول هسته برای دسترسی به هسته پیدا کرد.

استفاده از ضعف امنیتی در ماژول‌های معتبری که قبلاً در هسته سیستم‌عامل برگزاری شده‌اند و یا خود هسته سیستم‌عامل، می‌تواند راه دسترسی به هسته سیستم‌عامل باشد. برای این کار می‌باید ماژول هسته معتبری را که دارای اشکال امنیتی می‌باشد پیدا کرد. در ادامه می‌باید این اشکال را مورد بهره‌برداری[[45]](#footnote-45) قرار داده تا قطعه کد موردنظر در هسته اجرا شود. در این میان مکانیزم‌هایی در سیستم‌عامل وجود دارد تا بهره‌برداری از اشکالات امنیتی در برنامه‌ها و ماژول‌ها را غیرممکن کند؛ و البته روش‌هایی وجود دارد که مکانیزم‌های نام‌برده را از کار بی اندازد. خلاصه آنکه بهره‌برداری از اشکالات امنیتی در نرم‌افزارها خود مقوله‌ای بسیار وسیع و پیچیده است. به شکل مدام روش‌هایی برای از کار انداختن مکانیزم‌های جلوگیری از بهره‌برداری توسط نفوذ گران ایجاد می‌شود و متناسب با آن روش‌هایی برای از کار انداختن روش‌های جدید ایجادشده در سیستم‌عامل‌ها مورداستفاده می‌گیرد.

به علت محدودیت بسیار زیادی که در اجرای قطعه کدها در زمان بهره‌برداری از اشکالات وجود دارد، کدهای بسیار کوتاه و کوچکی می‌تواند در بهره‌برداری مورداستفاده قرار گیرد. درنتیجه این روش برای اجرای کل یک روت‌کیت استفاده نمی‌شود و لازم است کد اصلی روت‌کیت در یک ماژول مستقل قرارگرفته باشد و تنها روال غیرفعال سازی اعتبار سنجی به این روش اجرا می‌شود تا در ادامه بتوان ماژول روت‌کیت را بدون اشکال نصب و اجرا کرد.

برای نمونه می‌توان به کار Ralf Hund [13] اشاره کرد که با استفاده از برنامه‌نویسی برگشت گرا[[46]](#footnote-46) در بهره‌برداری از اشکالات موجود در هسته، به آن نفوذ کرده و مکانیزم اعتبار سنجی هسته را غیرفعال می‌کند.

### غیرفعال سازی روال‌های بررسی دوره‌ای هسته

به‌مانند روال‌های اعتبار سنجی ماژول، روال‌های بررسی دوره‌ای هسته نیز در خود هسته قرار دارد. البته این امر غیرقابل‌اجتناب است به دلیل آنکه برای بررسی تصویر هسته نیاز به دسترسی به حافظه آن هست و لازمه آن اجرا کد بررسی در خود هسته است. به همین دلیل غیرفعال سازی خود روال‌های بررسی دوره‌ای هسته نیز احتیاج به اجرا کد در سطح هسته سیستم‌عامل دارد.

برای اجرا کد غیرفعال سازی روال‌های بررسی دوره‌ای هسته می‌توان از روش توضیح داده‌شده در غیرفعال سازی اعتبار سنجی استفاده کرد، یعنی بهره‌برداری از اشکال امنیتی در هسته سیستم‌عامل و یا ماژول‌های معتبر موجود در هسته. به دلیل آنکه روال‌ها بررسی دوره‌ای در بازه‌های زمانی، در حد چند دقیقه عمل می‌کنند، نیازی به غیرفعال سازی آن‌ها در مرحله ابتدایی قبل از اجرا ماژول اصلی نیست. روال‌های غیرفعال سازی می‌تواند در خود ماژول اصلی روت‌کیت وجود داشته باشد و پس از بارگزاری آن در هسته روال‌های غیرفعال سازی اجرا شوند.

روال‌های بررسی دوره‌ای هسته از مکانیزم‌های امنیتی مبتنی بر پیچیده سازی کد برای محافظت از خود استفاده می‌کنند. روال‌های امنیتی مبتنی بر پیچیده سازی نمی‌توانند تضمینی برای حفاظت صد در صد در اختیار قرار دهند. تنها نفوذ و از کار انداختن را دشوار و زمان‌بر می‌کنند ولیکن این امر را غیرممکن نمی‌سازند. راه‌کاری که برای پوشش این ضعف مورداستفاده قرار می‌گیرد، بروز رسانی مدام روال‌های پیچیده سازی است. با این روش عمر روال‌های دور زدن مکانیزم‌های پیچیده سازی موجود کم می‌شود و همچنین نسبت زمان مفید بودن یک‌روال پیچیده سازی به غیرقابل استفاده بودن آن بیشتر خواهد شد.

در سری مقالاتی که در سایت uninformed.org [2] منتشرشده است شرحی کامل از نحوه عملکرد نسخه‌های مختلف روال‌های بررسی دوره‌ای هسته (Patch guard) بیان‌شده و همچنین نحوه دور زدن هریک از آن‌ها توضیح داده‌شده است.

## استفاده مجازی‌سازی در غیرفعال سازی روت‌کیت‌ها

با مرور مباحث مطرح‌شده در مورد روت‌کیت‌ها و روال‌های موجود برای جلوگیری از عملکرد آن‌ها می‌توان نتیجه گرفت هیچ‌یک از روال‌های موجود فعلی توانایی مقابله کامل و بی‌نقصی در برابر روت‌کیت‌ها ندارند. وجود اشکالات امنیتی در هسته و یا ماژول هسته‌ای که دیگر شرکت‌های تولید سخت‌افزار و یا نرم‌افزار آن‌ها را تولید می‌کنند، می‌تواند منجر به دسترسی به حافظه کدهای مکانیزم‌های امنیتی در سیستم‌عامل شده و درنتیجه می‌توان آن‌ها را از کار انداخت. وجود اشکالات امنیتی را نمی‌توان به شکل کامل از بین برد و درنتیجه می‌باید روشی برای عدم دسترسی به حافظه کدهای مکانیزم‌های امنیتی ایجاد کرد. مجازی‌سازی می‌تواند راه‌حلی مناسب برای جلوگیری از دسترسی به حافظه کدهای مکانیزم‌های امنیتی باشد.

یکی از اصول بنیادی در مجازی‌سازی این است که ماشین مجازی دیدی نسبت به مجازی بودن خود ندارد و نباید نحوه اجراشدن آن در ماشین مجازی با ماشین واقعی فرقی داشته باشد. برای احراز این اصل می‌باید سیستم مجازی‌سازی تمام دسترسی‌های ماشین‌های مجازی ایجادشده در یک سیستم را به‌تمامی منابع موجود کنترل و مدیریت کند. منابعی مانند حافظه اصلی، پردازنده و کلیه سخت‌افزارهای موجود به نحوی کنترل و مدیریت می‌شوند که برای ماشین مجازی این‌طور به نظر می‌رسد که تنها خودش به منابع موجود دسترسی دارد.

این سطح از کنترل در دسترسی به منابع سیستم ازجمله حافظه و پردازنده، در روال‌های محافظتی پردازنده (مد کاری محافظت‌شده و مد بلند) وجود ندارد. برای نمونه دسترسی به‌تمامی قسمت‌های حافظه برای کدی که در حالت ممتاز اجرا می‌شود وجود دارد و هیچ‌گونه کنترلی و ممیزی نمی‌شود برای روی آن اعمال کرد. به عبارتی درست‌تر هر دسترسی به هسته سیستم‌عامل منجر به دسترسی به‌تمامی نواحی حافظه ازجمله حافظه کدهای امنیتی در سیستم می‌گردد؛ اما اگر سیستم‌عامل و برنامه‌های کاربردی در درون یک ماشین مجازی اجرا شوند می‌توان دسترسی به نواحی از حافظه که کدهای امنیتی محافظت‌کننده در آن قرار دارند را محدود کرد و مانع آن شد که این روال‌ها به شکل ناخواسته از کار بی افتند.

به‌کارگیری مجازی‌سازی در جهت حفاظت از هسته سیستم‌عامل با به‌کارگیری متداول مجازی‌سازی متفاوت است. در به‌کارگیری متداول مجازی‌سازی هدف آن است که منابع و امکانات سخت‌افزاری یک سیستم، به شکلی در اختیار قرار گیرد که به‌مانند چند سیستم کاملاً مجازاً بتوان از آن استفاده کرد؛ اما تنها هدف از استفاده مجازی‌سازی در جهت حافظت، کنترل کامل سیستم است. تفاوت در اهداف استفاده باعث می‌شود نحوه استفاده نیز تغییر کند. در کاربرد اشتراک‌گذاری منابع، رویه‌ها و کدهای مجازی‌سازی بسیار پیچیده و به نسبت زیاد می‌باشند و به همان نسبت سربار زیادی را بر روی سیستم قرار می‌دهند. به‌عکس در حالتی که برای حفاظت از سیستم مورداستفاده قرار می‌گیرد رویه و کدهای مجازی‌سازی به نسبت کم و ساده پیاده‌سازی می‌شود و درنتیجه سربار نسبتاً کمی بر روی سیستم ایجاد می‌کنند.

مجازی‌سازی را می‌توان مستقل از سخت‌افزار و یا مبتنی بر سخت‌افزار انجام داد. در حالتی که مجازی‌سازی به شکل نرم‌افزاری صورت گیرد سربار بسیار زیادی برای روی سیستم ایجاد می‌کند که عملاً مجازی‌سازی کاربردی و قابل توجیه در این صورت میسر نیست. تنها مزیت این حالت این است که می‌توان مستقل از سخت‌افزار ماشین مجازی ایجاد کرد. برای نمونه می‌توان بر روی سیستم‌هایی با معماری ARM کدهای ماشین‌هایی با معماری x86 را اجرا کرد. به این حالت به اصلاح شبیه‌سازی [[47]](#footnote-47)گفته می‌شود. به علت اینکه در این روش تمامی اعمال صورت گرفته در یک سیستم را به شکل کامل می‌توان از طریق نرم‌افزار شبیه‌سازی دنبال کرد از این روش برای توسعه و عیب‌یابی نرم‌افزارهای سیستمی مانند سیستم‌عامل استفاده می‌کنند. پروژه نرم‌افزاری Bochs یکی معروف‌ترین نرم‌افزارهای این دسته می‌باشد.

مجازی‌سازی سخت‌افزاری مبتنی بر پردازنده است. پردازنده‌ها نسل حاضر دستورات و ساختمان داده‌هایی در اختیار قرار می‌دهند که می‌توان به کمک آن‌ها ماشین مجازی ایجاد کرد. در اینجا نیز نیاز به نرم‌افزار هست اما در این حالت، تنها کافی است، نرم‌افزار ساختمان داده‌های مربوطه در پردازنده را مقداری دهی کند و روال‌های سرویس‌دهی به رویدادهای تعریف‌شده برای مجازی‌سازی را پیاده‌سازی نماید. به این دسته از نرم‌افزارها ناظر ماشین مجازی (Virtual Machine monitor) گفته می‌شود. کارایی در این حالت بسیار بیشتر از مجازی‌سازی نرم‌افزاری هست و تنها در این حالت است که سربار مجازی‌سازی به حدی کم است که این عمل را قابل توجیه می‌کند. ولیکن برعکس مجازی‌سازی نرم‌افزاری در این حالت مجازی‌سازی وابسته به معماری بوده و نمی‌شود کدهای معماری دیگری را بر روی پردازنده با معماری متفاوت اجرا کرد. از این دسته نرم‌افزارهایی مانند VMware و XEN و KVM را می‌توان نام برد.

# **کارهای انجام‌شده در به‌کارگیری مجازی سازی در امنیت**

در کار Mads Bergdal و [3]Trond Arne Sornby بررسی یکپارچگی هسته سیستم‌عامل لینوکس موردبررسی قرارگرفته است. راه‌حل آن‌ها به شکل مشتری خدمت‌گزار ایجادشده است. قسمت مشتری در ماشین اصلی و قسمت خدمت‌گزار در ماشین‌های مجازی نصب می‌گردد. بستر مجازی‌سازی استفاده‌شده، برنامه XEN است. برنامه XEN یک ناظر ماشین مجازی است که خود دارای یک هسته کوچک است و می‌تواند مستقلاً و بدون نیاز به سیستم‌عامل، در درون سیستم اجرا شود. این برنامه دودسته ماشین مجازی را در خود اجرا می‌کند. ماشین مجازی در دامنه با دسترسی بالا که تنها یک ماشین مجازی بوده و تنها همین ماشین مجازی به‌طور مستقیم با سخت‌افزار در ارتباط است. و دیگر ماشین مجازی‌ها در دامنه دسترسی پایین اجرا می‌شوند و ارتباط آن‌ها با سخت‌افزار به‌واسطه ماشین مجازی موجز دار است.

در کار انجام‌شده برنامه‌ای که دستورات و تنظیمات را از کاربر گرفته و به سیستم‌های هدف می‌رساند در ماشین مجازی مجوز دار نصب می‌شود و برنامه که تنظیمات را دریافت و عملیات بررسی یکپارچگی هسته سیستم‌عامل را بر عهده دارد در ماشین‌های مجازی با دسترسی پایین اجرا می‌شوند. این برنامه‌ها گزارش‌های خود را به برنامه موجود در ماشین با دسترسی بالا می‌فرستند.

نقطه قوت کار انجام‌شده بررسی کامل و بدون نقص یکپارچگی سیستم است و نقطه‌ضعف اصلی آن وابسته بودن به بستر XEN است که خود باعث کندی سیستم شده و نمی‌توان یک سیستم‌عامل را بدون نیاز به یک سیستم دیگر موردحمایت قرارداد.

در کار دیگر که Min Zhu و [6]Miao Yu با عنوان به‌کارگیری مجازی‌سازی در نظارت امنیتی برای محافظت مستقل از سکو [[48]](#footnote-48) انجام داده اند. از ماشین مجازی برای محافظت از ورودی و خروجی‌ها و جلوگیری از تغییرات با ریسک بالا در تنظیمات پردازنده توسط ماشین مهمان، استفاده می‌شود. VMM طراحی‌شده بر اساس کد روت‌کیت تحقیقاتی، قرص آبی[[49]](#footnote-49) ایجادشده است. حافظه VMM برای ماشین مهمان غیرقابل رؤیت است و درنتیجه از کار انداختن و یا دور زدن آن توسط برنامه مهمان امکان‌پذیر نیست. رفتار نظارتی این پروژه قابل تنظیم است و می‌توان با آن رفتارهای خطرناک ممکن، مانند تغییر در ثبات‌های کنترلی و حساس، وقفه‌های اشکال‌زدایی و دسترسی به مکان‌های خاص حافظه را ره‌گیری کرد.

نقاطت قوت:

* استفاده از VMM کوچک و مستقل از سیستم‌عامل، که درنتیجه سربار کمی بر روی کل سیستم می‌گزارد
* چند سکویی توسعه داده‌شده است به معنی این‌که ماشین مجازی مهمان می‌تواند هر سیستم‌عامل باشد اعم از لینوکس و دیگر سیستم‌عامل‌های مبتنی بر unix و یا ویندوز.

نقاطت ضعف:

* به علتی که طراحی آن به شکل است که سیستم‌عامل ماشین مجازی می‌تواند هر نوعی باشد، نمی‌تواند نقاط حساس سیستم‌عامل‌ها را کنترل کند.
* به علتی که طراحی آن به شکل است که سیستم‌عامل ماشین مجازی می‌تواند هر نوعی باشد، امکان بررسی یکپارچگی سیستم‌عامل‌های وجود ندارد.

در کار Xianxian Li و [9]Changhui Jiang با استفاده از مجازی‌سازی سخت‌افزاری یک سیستم نظارتی ایجاد کرده‌اند که رویدادهای امنیتی زیر را می‌تواند ره‌گیری کرده و گزارش دهد:

* ره‌گیری فراخوانی‌های هسته و به دست آوردن اطلاعات فرایند فراخواننده. برای این کار مقدار ثبات SYSENTER\_EIP\_MSR را که در حالت معمول آدرس تابع توزیع‌کننده فراخوانی‌های هسته است، با عدد 0xffffff مقداردهی می‌شود. این مقدار آدرسی غیر معتبر است و درنتیجه یک رویداد نقص صفحه در هر بار فراخوانی هسته در سیستم ایجاد می‌شود. در VMX می‌توان برای نقص صفحه رویداد خروج تعریف کرد. در روال راه‌اندازی نقص صفحه بررسی می‌شود که اگر مقدار آدرس خواسته‌شده برابر با مقدار 0xfffffff باشد یک فراخوانی هسته ثبت می‌شود و بعد از به دست آوردن اطاعات کافی از این فراخوانی مقدار آدرس درست آن در ثبات اشاره‌گر آدرس اجرا قرار می‌گیرد و کنترل به ماشین مجازی بازمی‌گیرند.
* ره‌گیری تعویض متن فرایند. تعویض متن مابین فرایند با تغییر فهرست صفحه‌ها صورت می‌گیرد و برای تغییر فهرست صفحه باید مقدار CR3 تغییر کند. در VMX این امکان وجود دارد برای تغییر CR3 رویداد تعریف شود. با دریافت رویداد تغییر CR3 امکان تعیین تعویض متن فرایندهای نیز به وجود می‌آید.
* به دست آوردن اطلاعات فرایند جاری. در سیستم‌عامل لینوکس در حین تعویض متن مقدار ESP آدرس از حافظه را در خود ذخیره کرده است که اطلاعاتی در مورد فرایند جاری را در خود نگهداری می‌کند. این اطلاعات برای نتیجه‌گیری از اطاعات به‌دست‌آمده از روش‌های قبلی بسیار لازم و مفید است.

مجموع اطلاعات به یک جمع‌آوری‌کننده اطلاعات فرستاده می‌شود و نتایج بررسی‌شده و درصورتی‌که مخاطرات امنیتی ملاحظه شد به کاربر گزارش می‌دهد.

این پروژه بر روی KVM به همراه QEMU اجراشده است.

نقاط قوت:

* استفاده درست از امکانات VMX برای جمع‌آوری اطلاعات سیستم مهمان
* امکان تشخیص فرایندها مخفی‌شده و همچنین ارتباطات مخفی‌شده شبکه در سیستم

نقاط ضعف:

* عدم پوشش ورودی و خروجی سیستم
* عدم استفاده از یک VMM کوچک و سریع و نیازمند بدون به یک سیستم‌عامل به غیر سیستم‌عامل هدف برای انجام کار

در کار Arvind Seshadri و [7]Mark Luk یک VMM کوچک توسعه داده‌شده است که قابلیت‌های زیر را برای ایجاد امنیت دارا است:

* کنترل مدخل فراخوانی‌های هسته برای اطمینان از اینکه سرویس‌های هسته قلابHook) ) نشده باشند.
* محافظت از تصویر هسته در حافظه با استفاده از صفحه‌بندی تودرتو که در هر دو فنّاوری Inetl-VT و AMD-V وجود دارد. در این حالت پس آنکه ترجمه آدرس منطقی به آدرس فیزیکی در ماشین مجازی انجام شد بار دیگر یک ترجمه از آدرس به‌دست‌آمده به آدرس فیزیکی به‌وسیله فهرست صفحه‌ای که توسط VMX تعریف‌شده صورت می‌گیرد و سپس بر روی خطوط آدرس قرار می‌گیرد.
* جلوگیری از امکان دسترسی مستقیم و بدون کنترل به حافظه به‌واسطه ای DMA.

در این کار برای که محدوده هسته و ماژول‌های آن به‌درستی به دست آید می‌باید تغییرات در کد هسته سیستم‌عامل ایجاد شود. این کار برای سیستم‌عامل Linux انجام‌شده و کد هسته آن در این تحقیق به مقدار ناچیزی تغییر می‌کند.

نقطه قوت:

* عدم نیاز به سیستم‌عامل دیگر برای ایجاد VMM
* کنترل کل حافظه هسته سیستم‌عامل به نحوی امکان هرگونه تغییر و ایجاد وصله بر روی آن وجود ندارد.
* جلوگیری از دسترسی بی‌قید شرط به حافظه توسط سخت‌افزار.

نقطه‌ضعف:

* کاهش کارایی به خاطر استفاده از صفحه‌بندی تودرتو
* نیاز به تغییر در کد هسته سیستم‌عامل

در کار Takahiro Shainagawa و [4]Hideki Eiraku که یکی از کامل‌ترین کارهای انجام‌شده در این حوزه است، یک VMM مستقل از سیستم‌عامل توسعه داده‌شده که حدود 20 هزار خط کد منبع آن است. نام این پروژه Bitvisor است که به روش دست‌به‌دست [[50]](#footnote-50)یک ماشین مجازی ایجاد می‌کند که سیستم‌عامل را در آن بار گزاری می‌کند. درروش دست‌به‌دست دسترسی به سخت‌افزار بدون دخالت مستقیم VMM صورت می‌گیرد و تنها VMM واسط ارتباطات مابین ماشین مجازی با سخت‌افزاراست. استفاده از روش دست‌به‌دست این امکان را ایجاد می‌کند که بر ورودی و خروجی‌های نظارت اعمال شود و رفتارهای خطرناک ره‌گیری شود. به علتی که مدیریتی بر روی دسترسی به سخت‌افزار اعمال نمی‌شود درنتیجه تنها یک ماشین مجازی بر روی سیستم امکان کار کردن دارد.

مشکلی که در این حالت به موجود می‌آید این است که دسترسی به حافظه برای سخت‌افزارها اغلب به واسط DMA صورت می‌گیرد و پردازنده از این دسترسی خبردار نمی‌شود. برای حل این مسئله روشی که در این مقاله استفاده‌شده است این است که با استفاده از صفحه‌بندی تودرتو یک حافظه مجازی برای ماشین مجازی کرده است. در ادامه چون DMA تنها با حافظه فیزیکی در ارتباط است و قسمتی از حافظه فیزیکی که DMA با آن کار می‌کند را از دسترس ماشین مجازی خارج می‌کند و هر بار با اتمام کار DMA اطلاعات آن بر روی قسمتی از حافظه فیزیکی که برای ماشین مجازی قابل‌دسترسی است کپی می‌کند.

در این پروژه ورودی و خروجی بر روی دیسک نیز رمزنگاری می‌شود.

نقاط قوت:

بستر کامل برای تولید راه‌کارهای مبتنی بر مجازی‌سازی می‌تواند قرار گیرد.

سربار کمی بر روی سیستم قرار می‌دهد.

DMA را بدون استفاده از امکانات سخت‌افزاری مانند مکانیزم DEV می‌تواند کنترل کند.

نقاط ضعف:

ملاحظاتی برای تشخیص و یا جلوگیری از روت‌کیت‌ها ندارد.

یکپارچگی سیستم را بررسی نمی‌کند.

# معرفی روش

در پروژه انجام‌شده هدف استفاده از مجازی‌سازی سخت‌افزاری برای محافظت تنها بخشی از سیستم‌عامل (و نه کل سیستم‌عامل) است که وظیفه محافظت از هسته در برابر نفوذ را دارد. ایده اصلی پروژه این است که برای محافظت از هسته سیستم‌عامل در برابر نفوذ لازم نیست کل سیستم مراقبت شود و فقط تنها کافی است که مکانیزم‌های امنیتی موجود را محافظت کنیم تا به آن‌ها نفوذ نشود و از کار نیافتند و محافظت از کل سیستم را به عهده مکانیزم امنیتی موجود بگزاریم. فواید این کار را در ادامه برمی‌شماریم:

* نیاز به توسعه VMM بسیار کوچک و ساده دارد و درنتیجه امکان خطای کمتر و امنیت بیشتر میسر می‌شود.
* ایجاد سربار بسیار کم بر روی سیستم.
* نیاز به‌روزرسانی درازای هر بار تغییر روال‌های نفوذ نیست چون روال محافظتی موجود در سیستم‌عامل خود را بروز رسانی می‌کنند و این پروژه تنها از آن‌ها محافظت می‌کنند.
* نیازی به تغییر در هسته سیستم‌عامل ندارد و درنتیجه می‌توان بر روی سیستم‌عامل منبع بسته نیز مانند ویندوز از آن‌ها استفاده کرد.
* حوزه کار و عملیات محدود و مشخص وعدم تداخل با برنامه امنیتی نصب‌شده در سیستم مانند آنتی‌ویروس‌ها و دیوارهای آتش.
* می‌توان با تنظیماتی از دیگر ماژول مهم سیستم مانند برنامه‌های رمزنگاری و دیگر برنامه امنیتی نیز محافظت کرد.

## روش‌های محافظت از مکانیزم‌های امنیتی

به دو روش می‌توان از کد مکانیزم‌های امنیتی محافظت کرد:

1. استفاده از روش صفحه‌بندی تودرتو در VMM و کنترل دسترسی برای حافظه روال‌های محافظتی هسته سیستم‌عامل
2. تعریف یک نقطه شکست[[51]](#footnote-51) سخت‌افزاری برای دسترسی به حافظه در ماشین مجازی برای محل حساس روال‌های محافظتی.

عیب روش اول سربار زیاد آن است. به دلیل آنکه یک مرحله به مراحل ترجمه آدرس منطقی به فیزیکی اضافه می‌شود کارایی را کاهش می‌دهد؛ و مزیت آن کنترل کل حافظه روال‌های امنیتی است. عیب روش دوم، عدم پوشش کل حافظه کد روال‌هاست و تنها می‌تواند چهار بایت از حافظه را بررسی کند و درنتیجه در این حالت باید حساس‌ترین نقطه کد را انتخاب کرد و آنجا را مورد محافظت قرارداد. مزیت روش دوم سربار بسیار کم این روش است که تقریباً می‌توان گفت سرباری بر روی سیستم نخواهد گذاشت.

در این تحقیق به خاطر سربار کم از روش دوم استفاده‌شده است که البته از روش اول نیز می‌توان برای محافظت جامع‌تر و در حالاتی که هزینه سرباز قابل‌تحمل است استفاده کرد.

## نحوه پیدا کردن محل حساس روال‌های امنیتی

برای این کار می‌توان با بررسی روش‌های دور زدن موجود، محلی که آن‌ها با اعمال تغییرات روال‌های دور زدن را از کار می‌اندازند پیدا کرد و آنجا محافظت کرد. در این تحقیق از محلی در کار جلایری [15] در پروژه خود به کار گرفته استفاده‌شده است.

## نحوه تعریف نقطه شکست سخت‌افزاری

در معماری IA-32 و Intel 64 امکانات اشکال‌زدایی و نظارت بر کارایی ایجادشده است. این امکان اجازه اشکال‌زدایی برنامه‌ها در دسترسی به محل مشخصی از حافظه و یا آدرس خاصی از درگاه ورودی و خروجی را فراهم می‌کند؛ و همچنین می‌توان نوع دسترسی و اندازه دسترسی را می‌توان کنترل کرد.

هشت ثبات اشکال‌زدایی عملیات اشکال‌زدایی را در پردازنده کنترل می‌کنند. این هشت ثبات به شرح زیر است.

DR0-DR3: چهار ثبات 32-64 بیتی که آدرس محل نقطه شکت را در خود نگهداری می‌کنند.

DR6: در صورت ایجاد رویداد اشکال‌زدایی در سیستم، اطلاعات رویداد ایجادشده در این ثبات نگهداری می‌شود. این اطاعات شامل شماره ثبات نقطه شکستی که باعث رویداد شده و نوع رویداد اشکال‌زدایی ایجادشده می‌شود.

DR7: تنظیمات اشکال‌زدایی در این ثبات صورت می‌گیرد. در ثبات می‌توان تعیین کرد که چه دسترسی به حافظه اعم از خواندن نوشتن و یا اجرا باعث ایجاد رویداد اشکال‌زدایی شود و یا می‌توان تعیین کرد که چه اندازه از دسترسی به حافظه منجر به رویداد شود. اندازه می‌تواند یکی از چهار مقدار 1-2-4-8 بایتی را داشته باشد. در این ثبات همچنین تعیین می‌شود که نقطه شکست‌ها عمومی باشند و یا خیر. درصورتی‌که نقطه شکستی عمومی باشد با تعویض متن در فرایندها تغییر نمی‌کند. در این صورت برای همه فرایندها عمومیت دارد.

## الگوریتم

1. محل آسیب‌پذیر روال محافظتی هسته را پیدا می‌کنیم.
2. یک ماژول هسته در سیستم برگزاری می‌شود و در روتین شروع به کار آن عملیات زیر انجام می‌شود.
   1. محل آسیب‌پذیر از برنامه کاربردی دریافت می‌شود.
   2. پردازنده وارد حالت کاری VMX می‌شود.
   3. ساختمان داده VMCS مقداردهی می‌شود. این مقداردهی به شکلی است تنظیمات ماشین جاری برای ماشین مجازی تعیین‌شده و برای VMM آدرس روالی تعیین می‌شود که آن روال در خود ماژول هسته جاری نوشته‌شده است. مدخل ورودی ماشین مجازی آدرس پایان روال شروع به کار ماژول هسته جاری قرار می‌گیرد.
   4. یکی از ثبات‌های نقطه شکست در سیستم مجازی، با مقدار آدرس محل آسیب‌پذیر پر می‌شود.
   5. برای رویداد اشکال‌زدایی در ماشین مجازی و دسترسی به نقطه شکست تعریف‌شده، یک رویداد خروج به vmm در VMCS تعریف می‌شود.
   6. ماشین مجازی اجرا می‌شود.
3. در این مرحله سیستم‌عامل در درون یک ماشین مجازی در حال کار است که البته هیچ‌گونه تغییری جز تعریف یک نقطه شکست عمومی در آن انجام‌نشده است و درنتیجه سیستم‌عامل بدون هیچ‌گونه اشکالی به کار خود در ماشین مجازی ادامه می‌دهد.
4. در صورتی روال محافظتی از طریق رونوشت کردن نقطه حساس موردحمله قرار گیرد. رویداد خروج به VMM اجرا می‌شود.
5. به دو دلیل روال خروج VMM صورت می‌گیرد
6. به دلیل رونوشت بر روی روال محافظتی: در روال خروج به VMM آدرس دستور رونوشت کننده و همچنین طول آن به دست می‌آید. مدخل ورود به ماشین مجازی را به بعد از دستور جاری تغییر داده و ماشین مجازی را دوباره اجرا می‌کنیم.
7. به دلیل دسترسی به ثبات نقطه شکست: درصورتی‌که ماشین مجازی قصد نوشتن برای ثبات را داشته باشد مقدار آن خوانده‌شده و نگهداری می‌شود. درصورتی‌که ماشین مجازی قصد خواندن از روی ثبات را داشته مقدار نگهداری شده برای آن اعلام می‌شود. با این کار ماشین مجازی از مقدار واقعی آن خبردار نمی‌شود.

## ملاحظات الگوریتم پیشنهادی

* در سامانه‌های چندپردازنده‌ای باید برای هریک از پردازنده‌ها و هسته‌های پردازشی روال ایجاد ماشین مجازی انجام گیرد.
* در ماشین مجازی هدف تنها امکان استفاده از 3 ثبات نقطه شکست ممکن است.
* در این مسئله فرض شده است که روت‌کیت از وجود VMM خبر ندارد. به همین دلیل حافظه VMM در دسترس ماشین مجازی قرار دارد ولیکن برای حصول اطمینان بیشتر می‌توان به روش صفحه‌بندی تودرتو حافظه VMM را از دسترس ماشین مجازی خارج کرد.

# **بررسی میزان اثرگذاری بر روی کارایی سیستم**

برای بررسی میزان اثرگذاری بر روی کارایی سیستم از یک برنامه محک استاندارد به نام SPECint2006استفاده شد. این برنامه بر روی یک سیستم‌عامل ویندوز 7 با پردازنده corei7 اجرا گردید که نتیجه را در شکل شماره 18 مشاهده می‌شود.

شکل شماره 18

# **نتیجه‌گیری**

مجازی‌سازی سخت‌افزاری می‌تواند ضعف عمده مکانیزم‌های امنیتی موجود را پوشش دهد. با در نظر گرفتن میزان اثرگذاری بر روی سیستم می‌توان گفت نسبت فایده به هزینه خوبی حتی برای سیستم‌هایی با پردازش زیاد دارد. البته نباید از اثر، نحوه درست به کاری گیری و پیاده‌سازی این روش در سیستم، بر روی میزان کارایی روش چشم‌پوشی کرد.

درروش ارائه‌شده در این تحقیق اولویت بر روی کارایی ماشین مجازی بود. تا کمترین اثر را بر روی کارایی اصلی سیستم داشته باشد که البته فرض ناآشنا بودن حمله‌کننده به سیستم با این روش کمی ازنظر امنیتی کار را دچار اشکال می‌کرد که اگر از صفحه‌بندی تودرتو ترکیب شود این نقص نیز برطرف می‌شود اما کارایی روش مذکور نیز از دست می‌رود.

# منابع

|  |  |
| --- | --- |
| [1] | S. Jalayeri, “DriveCrypt,” [درون خطي]. Available: https://github.com/shjalayeri/DriveCrypt. |
| [2] | Skywing, “A Brief Analysis of PatchGuard Version 3,” http://uninformed.org/, 2007. [درون خطي]. Available: http://uninformed.org/index.cgi?v=8&a=5. |
| [3] | Mads Bergdal , Trond Arne Sørby, “Using virtual machines for integrity checking,” *Master thesis , UNIVERSITY OF OSLO Department of Informatics,* May 1st 2007. |
| [4] | Shinagawa, Takahiro and Eiraku, Hideki and Tanimoto, Kouichi and Omote, Kazumasa and Hasegawa, Shoichi and Horie, Takashi and Hirano, Manabu and Kourai, Kenichi and Oyama, Yoshihiro and Kawai, Eiji and others, “Bitvisor: a thin hypervisor for enforcing i/o device security,” در *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, 2009. |
| [5] | Becher, Michael and Dornseif, Maximillian and Klein, Christian N, “FireWire: all your memory are belong to us,” در *Proceedings of CanSecWest*, 2005. |
| [6] | M. Zhu, “VASP: virtualization assisted security monitor for cross-platform protection,” در *Proceedings of the 2011 ACM Symposium on Applied Computing*, 2011. |
| [7] | A. a. L. M. a. Q. N. a. P. A. Seshadri, “SecVisor: A tiny hypervisor to provide lifetime kernel code integrity for commodity OSes,” *ACM SIGOPS Operating Systems Review,* جلد 41, شماره ACM, p. 6, 2007. |
| [8] | de Oliveira, Daniela Alvim Seabra and Wu, S Felix, “Protecting kernel code and data with a virtualization-aware collaborative operating system,” در *Computer Security Applications Conference, 2009. ACSAC'09. Annual*, IEEE, 2009. |
| [9] | X. Li, “Vminsight: Hardware virtualization-based process security monitoring system,” در *Network Computing and Information Security (NCIS)*, 2011. |
| [10] | microsoft, “msdn.microsoft.com,” [درون خطي]. Available: https://msdn.microsoft.com/en-us/library/ms537361(v=vs.85).aspx. |
| [11] | microsoft, “An Introduction to Kernel Patch Protection,” microsoft, [درون خطي]. Available: https://blogs.msdn.microsoft.com/windowsvistasecurity/2006/08/12/an-introduction-to-kernel-patch-protection/. |
| [12] | R. Johndas, “Analysis of CVE-2014-4113 Windows privilege Escalation Vulnerability,” 2014. |
| [13] | R. Hund, “Return-Oriented Rootkits: Bypassing Kernel Code Integrity Protection Mechanisms,” در *USENIX Security Symposium*, 2009. |
| [14] | Popek, Gerald J., and Robert P. Goldberg, “Formal requirements for virtualizable third generation architectures,” *Communications of the ACM,* شماره 17.7, pp. 412-421, 1974. |
| [15] | ش. جلایری, “DriveCrypt,” 2014. [درون خطي]. Available: https://github.com/shjalayeri/DriveCrypt. |
| [16] | wikipedia, “Hardware-assisted virtualization,” [درون خطي]. Available: https://en.wikipedia.org/wiki/Hardware-assisted\_virtualization. |
| [17] | Intel, Intel® 64 and IA-32 Architectures Software Developer’s Manual (Combined Volumes: 1, 2A, 2B, 2C, 3A, 3B and 3C), Intel, 2014. |
| [18] | Intel, INTEL 80386 PROGRAMMER'S REFERENCE MANUAL 1986, Intel. |
| [19] | Prasad Dabak , Sandeep Phadke , Milind Borate, Undocumented Windows NT, Hungry Minds. |
| [20] | Greg Hoglund, James Butler, Rootkits: Subverting the Windows Kernel, Addison Wesley Professional. |
| [21] | David Solomon and Mark Russinovich, Windows Internals, Microsoft Press. |

­

1. ًRootkit [↑](#footnote-ref-1)
2. Privilege escalation [↑](#footnote-ref-2)
3. مراجع تمامی مطالب بیان شده در قسمت مدیریت حافظه و حافظت، کتاب ‌های راهنمای شرکت ّ [18]Intel [17] میباشد. [↑](#footnote-ref-3)
4. Real mode [↑](#footnote-ref-4)
5. Protect mode [↑](#footnote-ref-5)
6. Section [↑](#footnote-ref-6)
7. Page [↑](#footnote-ref-7)
8. Long mode [↑](#footnote-ref-8)
9. Segment translation [↑](#footnote-ref-9)
10. Segment selector [↑](#footnote-ref-10)
11. Page translation [↑](#footnote-ref-11)
12. Descriptors [↑](#footnote-ref-12)
13. Descriptor tables [↑](#footnote-ref-13)
14. Selectors [↑](#footnote-ref-14)
15. تمامی عکس های استفاده شده در قسمت مدیریت حافظه و خافظت از کتاب های راهنمای Intel [18] [17] گرفته شده است. [↑](#footnote-ref-15)
16. Granularity [↑](#footnote-ref-16)
17. DPL (Descriptor Privilege Level) [↑](#footnote-ref-17)
18. global descriptor table (GDT) [↑](#footnote-ref-18)
19. local descriptor table (LDT) [↑](#footnote-ref-19)
20. Table indicator [↑](#footnote-ref-20)
21. Requested privilege level [↑](#footnote-ref-21)
22. Expansion-Direction bit [↑](#footnote-ref-22)
23. Big bit [↑](#footnote-ref-23)
24. descriptor privilege level(DPL) [↑](#footnote-ref-24)
25. requestor's privilege level(RPL) [↑](#footnote-ref-25)
26. current privilege level(CPL) [↑](#footnote-ref-26)
27. ثبات‌های مدل - مخصوص REGISTERS (MSRs)MODEL-SPECIFIC دسته‌ای از ثبات‌ها هستند که به‌طور مخصوص در اختیار سیستم‌عامل و روال‌های اختصاصی (آن‌هایی که در سطح دسترسی صفر اجرا می‌شوند) قرار داده‌شده‌اند. این ثبات‌ها مواردی از قبیل الحاقات اشکال‌زدایی – شمارنده دیدبان کارایی – بررسی معماری ماشین – دامنه نوع حافظه- دستورات مخصوص پشتیبان (SYSENTER – SYSEXIT – SWAPGS و غیره) را کنترل می‌کنند. این ثبات‌ها از پردازنده‌های پنتیوم به بعد وارد معماری IA\_32 شده‌اند. این ثبات‌ها به‌وسیله دستورات RDMSR و WRMSR خوانده و نوشته می‌شوند. [↑](#footnote-ref-27)
28. Page Dictionary Entry [↑](#footnote-ref-28)
29. Page Table Entry [↑](#footnote-ref-29)
30. Supervisor level [↑](#footnote-ref-30)
31. مطالب گفته شده در بخش معماری سیستم عامل از کتاب های Undocumented Windows NT [19] و [20] Rootkits: Subverting the Windows Kernel و Windows internal [21] برداشت شده است. [↑](#footnote-ref-31)
32. Application Programming Interfaces (APIs) [↑](#footnote-ref-32)
33. System Service Dispatch Table (SSDT) [↑](#footnote-ref-33)
34. Dynamic-Link Library (DLL) [↑](#footnote-ref-34)
35. kernel-mode drivers [↑](#footnote-ref-35)
36. mainframes [↑](#footnote-ref-36)
37. Formal Requirements for Virtualizable Third Generation Architectures [↑](#footnote-ref-37)
38. trap-and-emulate [↑](#footnote-ref-38)
39. virtual-machine extensions [↑](#footnote-ref-39)
40. ٰ Virtual-machine monitors [↑](#footnote-ref-40)
41. Rootkit [↑](#footnote-ref-41)
42. firmware [↑](#footnote-ref-42)
43. bootkit [↑](#footnote-ref-43)
44. Privilege escalation [↑](#footnote-ref-44)
45. exploit [↑](#footnote-ref-45)
46. retrun-oriented [↑](#footnote-ref-46)
47. emulation [↑](#footnote-ref-47)
48. Virtualization assistd Security Monitor for Cross Platform Protection [↑](#footnote-ref-48)
49. Blue Pill [↑](#footnote-ref-49)
50. pass-through [↑](#footnote-ref-50)
51. ‌Breakpoint [↑](#footnote-ref-51)