پردیس علوم

دانشکده ریاضی، آمار و علوم کامپیوتر

مهندسی معکوس در حوزه طراحی نرم‌افزار

نگارنده

مرضیه اکبری‌فر

استاد راهنما: مهندس ابراهیم نقیب‌زاده مشایخ

پایان‌نامه برای دریافت درجه کارشناسی

در رشته علوم کامپیوتر

تیر ماه ۱۳۹۷

چکیده

مهندسی نرم‌افزار مرسوم در درجه اول بر تولید و طراحی نرم‌افزار جدید تمرکز دارد. با این حال بسیاری از برنامه‌نویسان کار کردن روی برنامه طراحی و توسعه داده شده توسط دیگران را ترجیح می‌دهند. مهندسی معکوس نرم‌افزار، فرایند ترجمه برنامه کامپایل شده به کد منبع و تحلیل کد حاصل است. بدون داشتن اطلاعات از درون یک برنامه، اضافه کردن ساختاری جدید به آن و یا ساختن برنامهای مشابه با آن دشوار است. از این رو یادگیری روش‌های مهندسی معکوس و استفاده از آن‌ها در حوزه نرم‌افزار کاربرد فراوان دارد. چالش مهندسی معکوس این است که در طی فرایند ترجمه قسمت زیادی از اطلاعات از بین می‌رود و به دست آوردن مجدد آن‌ها نیاز به تحلیل‌های مختلفی دارد. در این نوشتار به تعریف مهندسی معکوس در حوزه نرم‌افزار و کاربردهای آن می‌پردازیم. با وجود این که مهندسی معکوس شدیدا وابسته به نیروی انسانی است، ابزارهایی را برای کمک به برنامه‌نویس برای حل موانع و مشکلات معکوس‌سازی ارائه می‌دهیم.

پیشگفتار

در گذشته، فرایند مهندسی معکوس به خرید یک محصول و کالبدشکافی فیزیکی به منظور کشف اسرار طراحی آن خلاصه می‌شد. سپس از این اسرار برای طراحی و ساخت محصولات مشابه یا بهبودیافته استفاده شده و محصول مستقلی به تولید می‌رسید. اما ریشه مهندسی معکوس نرم‌افزار که امروزه استفاده می‌شود به اوایل دهه ۱۹۸۰ و تصمیم شرکت ای بی ام مبنی بر توقف انتشار کد منبع سیستم عامل رایانه‌های خود برمی‌گردد. کاربران تا آن زمان از کد منبع منتشر شده توسط ای بی ام برای تولید و انطباق سیستم عامل با نیازهای خود استفاده می‌کردند و این شرکت به واسطه BIOS (سیستم پایه‌ی ورودی و خروجی) رایانه‌ها که رابط بخش سخت‌افزاری و نرم‌افزاری آن‌ها بود، بازار را به طور کامل در اختیار خود نگه داشته بود. در این زمان شرکت کامپک که بزرگترین شرکت تولیدکننده رایانه‌های شخصی در ایالت تگزاس آمریکا بود، با استفاده از مهندسی معکوس بر BIOS (سیستم پایه‌ی ورودی و خروجی) رایانه‌های شرکت ای بی ام موفق شد اولین رایانه‌های شخصی سازگار با ای بی ام را وارد بازار کند. شاید بتوان این امر را یکی از مهم‌ترین زمینه‌های استفاده از مهندسی معکوس در حوزه نرم‌افزار دانست. در این نوشتار سعی می‌کنیم با مفهوم مهندسی معکوس و روش‌ها و ابزارهای مورد استفاده برای انجام آن آشنا شویم. در فصل اول مفاهیم مقدماتی مورد نیاز برای ادامه بحث را بررسی خواهیم کرد. در فصل دوم به مفاهیم نرم‌افزارهای سطح پایین و

زبان اسمبلی خواهیم پرداخت. در فصل سوم تعدادی از ابزارهای مورد استفاده در مهندسی معکوس نرم‌افزار و نقاط قوت آن‌ها را معرفی خواهیم کرد. در فصل چهارم مهندسی معکوس نرم‌افزار را از جنبه قانونی بررسی کرده و شروط محدود کننده برای نقض نکردن حق نشر محصول را بیان خواهیم کرد. در نهایت در فصل پنج خلاصه‌ای از مباحث نوشتار ارائه خواهد کرد.

مرضیه اکبری‌فر

تابستان ۹۷

فهرست مطالب

فصل ۱

مفاهیم اساسی

این فصل اطلاعاتی پایه‌ای درباره مهندسی معکوس و سایر مباحث مطرح شده در مقاله را ارائه می‌دهد. ابتدا تعریف و کاربردهای مختلف مهندسی معکوس در نرم‌افزار بیان شده و در ادامه رابطه‌ی آن با نرم‌افزارهای سطح پایین بررسی می‌شود. در نهایت مقدمه‌ای بر روند و ابزارهای مورد نیاز برای انجام مهندسی معکوس ارائه می‌شود.

۱.۱ مهندسی معکوس چیست؟

مهندسی معکوس فرایند استخراج دانش یا نقشه‌های طراحی از هر چیز ساخته شده است. این مفهوم مدت‌ها پیش از رایانه‌ها و فناوری کنونی وجود داشته و قدمت آن احتمالاً به زمان انقلاب صنعتی بازمی‌گردد. این عمل بسیار شبیه به یک تحقیق علمی است که در آن محقق سعی دارد طرح اتم یا ذهن انسان را به دست آورد، با این تفاوت که در تحقیقات علمی پدیده‌های طبیعی بررسی می‌شوند

ولی در مهندسی معکوس عموماً ساخته‌های دست بشر مورد مطالعه قرار می‌گیرند. در گذشته نه چندان دور، مهندسی معکوس به عنوان سرگرمی افراد تلقی می‌شد. به عنوان مثال باز کردن اجزای درونی یک رادیو به منظور کشف نحوه عملکرد آن و یا ساختن محصول مشابه یا بهتر، مهندسی معکوس است. گاهی اوقات اطلاعات مورد نیاز از دست رفته است و یا متعلق به کسی است که تمایلی به اشتراک‌گذاری آن‌ها ندارد. مهندسی معکوس به دست آوردن این دانش گمشده می‌باشد.

۲.۱ مهندسی معکوس نرم‌افزار

امروزه نرم‌افزار یکی از پیچیده‌ترین و در عین حال جالب‌ترین فناوری‌های اطراف ما است. همانند مهندسی نرم‌افزار، مهندسی معکوس نرم‌افزار یک عمل کاملاً مجازی بوده و ابزار مورد نیاز آن، یک پردازنده و ذهن خلاق می‌باشد. یادگیری این عمل همچنین نیاز به درکی کلی از رایانه‌ها و تولید نرم‌افزار دارد اما مانند سایر علوم ارزشمند، پیش‌نیاز واقعی آن کنجکاوی و تمایل به یادگیری است.

مهندسی معکوس نرم‌افزار مانند باز کردن «جعبه» برنامه و نگاه کردن به اجزای آن و ترکیبی از چندین هنر برنامه‌نویسی، حل معما، تجزیه و تحلیل منطقی و کدخوانی می‌باشد.

۳.۱ کاربردهای معکوس‌سازی

می‌توان گفت مهم‌ترین کاربرد مهندسی معکوس تولید محصولات رقابتی است. البته این روش چندان متداول نیست و دلیل اصلی آن پیچیده بودن و هزینه‌ی بالای تولید نرم‌افزار با این روش است. به طور کلی دو دسته کاربرد برای مهندسی معکوس وجود دارد: مربوط به امنیت[1] و مربوط

[1] Security-related reversing

به تولید نرم‌افزار[1] در ادامه به شرح این دو کاربرد می‌پردازیم.

۱.۳.۱ معکوس‌سازی مربوط به امنیت

معکوس‌سازی از جنبه‌های گوناگون به امنیت مربوط می‌باشد. این فرایند در رمزنگاری و ارزیابی سطح امنیت یک محصول رمزنگاری شده، تولید نرم‌افزارهای ضد ویروس، بررسی نحوه عملکرد بدافزارها و حتی شناخت لایه‌های امنیتی و گذر از آن‌ها به وسیله رخنه‌گران به کار گرفته می‌شود. همچنین برای بازبینی امنیتی، دسترسی به کدهای حساس و تحلیل بخش‌های تشکیل‌دهنده محصول از معکوس‌سازی استفاده می‌شود.

۲.۳.۱ معکوس‌سازی مربوط به تولید نرم‌افزار

معکوس کردن به طور قابل ملاحظه‌ای برای تولیدکنندگان نرم‌افزار مفید است. تولیدکنندگان می‌توانند روش‌های معکوس‌سازی را به منظور تعامل با نرم‌افزارهای فاقد مستندات کافی به کار گیرند. همچنین برای تعیین کیفیت کد شخص ثالث و استخراج اطلاعات ارزشمند از محصول به منظور بهبود کیفیت آن نیز می‌توان از معکوس سازی استفاده کرد. در ادامه به شرح کاربردهای ذکر شده می‌پردازیم.

قابلیت همکاری در نرم‌افزارهای اختصاصی

اغلب اوقات مهندسان هنگام کار با نرم‌افزار یا رابط برنامه‌نویسی کاربردی متوجه کافی نبودن مستندات می‌شوند. حتی اگر زمان زیادی صرف نوشتن مستندات شده باشد، ممکن است سوالی

[1] Software development related reversing

برای شخص به وجود آید که جواب آن در مستندات موجود نباشد. تماس با فروشنده و رفع مشکل ممکن است هزینه‌بر و یا غیرممکن باشد. مهندسین مسلط به روش مهندسی معکوس می‌توانند این دسته از مشکلات را در زمان کم و با هزینه نسبتاً کم رفع کنند.

تولید نرم‌افزارهای رقابتی

همان‌طور که اشاره شد، تولید محصول رقابتی یکی از مهم‌ترین کاربردهای مهندسی معکوس در صنعت است. البته نرم‌افزار محصولی پیچیده است و معکوس سازی همه‌ی کد آن غیرمنطقی است. بنابراین روش راحت‌تر و متداول استفاده از بخش‌های از پیش ساخته شده کدهای شخص ثالث و تغییر دادن آن‌ها به منظور تولید محصول مطلوب است. ساخت قسمت‌های منحصر به فرد یا راحت نرم‌افزار به صورت مستقل و از ابتدا در اغلب موارد روش راحت‌تری است. این روش باید تنها برای بخش‌های بسیار پیچیده یا هزینه‌بر و با مجوز قانونی صورت گیرد. مهندسی معکوس با برآورد کوشش مورد نیاز برای تغییر سیستم به وضعیت دلخواه، امکان نوسازی نرم‌افزار را نیز فراهم می‌کند.

ارزیابی کیفیت و قابلیت اطمینان نرم‌افزار

نرم‌افزارهای متن‌باز با در اختیار قرار دادن کد منبع، این امکان را برای کاربران فراهم می‌کنند که قابلیت اطمینان و کیفیت محصول را ارزیابی کنند. در مقابل فروشندگانی هستند که کد منبع را منتشر نمی‌کنند و از خریدار می‌خواهند که به آن‌ها اعتماد کند. همان‌طورکه وارسی کد دودویی برنامه روشی مناسب برای ارزیابی آسیب‌پذیری و امنیت نرم‌افزار است، نمونه‌گیری از کد دودویی و بررسی آن راه‌حلی برای تخمین میزان کیفیت و قابلیت اطمینان آن است. به وضوح معکوس‌سازی به اندازه‌ی بررسی کد منبع اطلاعات را افشا نمی‌کند ولی استفاده از آن در بسیاری از موارد آموزنده

است.

۴.۱ نرم‌افزارهای سطح پایین

نرم‌افزار سطح پایین یانرم‌افزار سیستم، زیرساختی از دنیای نرم‌افزار شامل مترجم‌ها، اشکال‌زداها، سیستم عامل‌ها و زبان‌های برنامه‌نویسی سطح پایین مانند زبان اسمبلی است. این همان لایه‌ای است که تولیدکنندگان نرم‌افزار و برنامه‌های کاربردی را از سخت‌افزار فیزیکی جدا می‌کند. ابزارهای تولید نرم‌افزار برنامه‌نویس را از پیچیدگی زبان اسمبلی رهایی بخشیده و سیستم‌های عامل آن‌ها را از جزئیات سخت‌افزاری جدا کرده و در عین حال تعامل کاربر را با سیستم را آسان ساخته است. اگر فرایند ساخت نرم‌افزار را به صورت چند لایه در نظر بگیریم، در لایه پایین میلیون‌ها ترانزیستور وجود دارند که با سرعت زیادی دستورهای فرستاده شده از کاربر را پردازش می‌کنند. درگذشته، تنها روش نوشتن نرم‌افزار برای برنامه‌نویسان کار در این لایه بود. امروزه سیستم‌عامل‌های پیشرفته و ابزارهای تولید با هدف جداسازی برنامه‌نویس از جزئیات کار در سطح پایین وجود دارند. بنابراین بیشتر برنامه‌نویسان ترجیح می‌دهند از زبان‌های سطح بالا که فرمان‌های قابل فهم گرفته و اجرا می‌کنند، استفاده کنند. این ابزارها روند تولید نرم‌افزار را آسان کرده‌اند اما قدرت و تسلط برنامه‌نویس بر سیستم را نیز کاهش داده‌اند. فرمان ساده‌ای در سطح بالا ممکن است به میلیون‌ها خط کد در سطح پایین ترجمه شود. بنابراین برای استفاده از مهندسی معکوس درک جامعی از نرم‌افزار و برنامه‌نویسی سطح پایین نیاز است و برنامه‌نویس باید از هر چیزی که بین پردازنده و کد اتفاق می‌افتد مطلع باشد، زیرا جنبه‌های سطح پایین برنامه تنها چیزی است که مورد بررسی قرار می‌گیرد.

در ادامه جنبه‌هایی از سطح پایین که دانستن آن‌ها برای موفقیت در معکوس کردن الزامی است،

بررسی می‌شوند.

۱.۴.۱ زبان اسمبلی

پایین‌ترین سطح در زنجیره نرم‌افزار زبان اسمبلی است و هر عملی که در نرم‌افزار انجام می‌شود، باید به کد زبان اسمبلی تبدیل شود. این زبان ساختار و عملکردی وابسته به ماشین دارد. باید توجه داشت که زبان ماشین و زبان اسمبلی دو نحوه نمایش مختلف یک مفهوم هستند. زبان ماشین، زبان ذاتی و انحصاری رایانه است که هنگام طراحی سخت افزار رایانه تعریف می‌شود. زبان ماشین، شامل رشته‌ای از اعداد است و سبب می شود که رایانه عملیات اصلی مربوط به خود را در هر بار راه اندازی اجرا کند.

درک زبان ماشین برای انسان دشوار است. بنابراین برنامه‌نویسان به جای به کار بردن رشته ای از اعداد که رایانه بتواند به طور مستقیم آن‌ها را درک کند، از عبارت های مخفف شده ای شبیه زبان انگلیسی برای فهماندن عملیات ابتدایی به رایانه استفاده می‌کنند. این عبارت‌های مخفف شده و شبه انگلیسی، مبانی زبان اسمبلی هستند. هر دستور در زبان اسمبلی با عددی نمایش داده می‌شود که به آن کد عملیاتی[۱] می‌گویند. کد زبان ماشین[۲] یک توالی از کدهای عملیاتی و سایر اعدادی است که باعث انجام یک عمل می‌شود. پردازنده برای اجرای کدهای عملیاتی، ابتدا آن‌ها را یک به یک از حافظه برنامه فراخوانی می‌کند و سپس توسط واحد رمزگشای خود، آن ها را رمزگشایی و عملیات آن‌ها را انجام می دهد. زمانی که تولید دهنده کدی به زبان اسمبلی می‌نویسد، باید برای اجرا آن را به زبان اسمبلی تبدیل کند. این کار توسط اسمبلر صورت می گیرد. اسمبلر برنامه ای

[۱] Operation code

[۲] Object code

است که فایل متنی حاوی دستورات اسمبلی را خوانده و نمادهای آن را به کدهای زبان ماشین تبدیل می کند. عملیات برعکس اسمبلر که در معکوس‌سازی کاربرد فراوانی دارد توسط دیس‌اسمبلر[1] انجام می‌شود. این برنامه کد زبان ماشین را گرفته و کد زبان اسمبلی را که برنامه به زبان ماشین از آن ایجاد می‌شود تولید می‌کند.

از آنجا که زبان اسمبلی وابسته به سخت‌افزار و پردازنده است، برای ادامه توضیحات از معماری IA-۳۲ Intel استفاده می‌کنیم.

۲.۴.۱ مترجم‌ها

از زمانی که برنامه نویسان به استفاده از زبان‌های سطح بالا روی آوردند، مترجم‌ها نیز به عرصه رایانه وارد شدند. مترجم یک برنامه نرم‌افزاری است که کدی را که توسط برنامه‌نویس نوشته شده به کد دودویی (کد ماشین) تبدیل می‌کند. این فرایند باعث می‌شود تا کدها توسط پردازنده تشخیص داده شده و قابل درک و اجرا باشند.

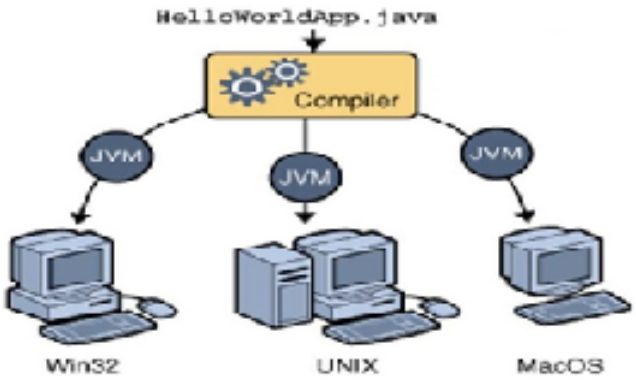
با توجه به نوع زبان سطح بالا، کد ماشین تولید شده ممکن است یک شی وابسته به ماشین باشد و مستقیماً توسط پردازنده رمزگشایی شود، یا به صورت بایت‌کد رمزگذاری شود که قابلیت حمل دارد.

بزرگترین مانع در رمزگشایی کد تولید شده توسط مترجم، بهینه‌سازی انجام شده روی کد است. اکثر مترجم‌ها با استفاده از روش‌های مختلف سعی در به حداقل رساندن اندازه کد و بهبود عملکرد اجرایی آن دارند. این کار معمولاً با جایگزینی عبارات پیچیده ریاضی انجام می‌شود که خوانایی کد را کم می‌کند.

[1] disassembler

۳.۴.۱ ماشین‌های مجازی و بایت‌کدها

مترجم‌های زبان‌های سطح بالا مانند جاوا به جای کد زبان ماشین، بایت‌کد تولید می‌کنند. برنامه‌های جاوا، قابلیت حمل در شبکه را دارند ۱.۱. این برنامه‌ها پس از ترجمه به دستوراتی تبدیل می‌شوند که برای ماشین مجازی جاوا[۱] قابل فهم است. این دستورات بایت کد نام دارند. ماشین مجازی جاوا بایت کد را به کد ماشین که برای سخت‌افزار کامپیوتر قابل فهم است، تبدیل می‌کند. بنابراین بایت کدها را می‌توان در هر جای شبکه، روی هر کارسازی که یک ماشین مجازی جاوا دارد، اجرا کرد.



شکل ۱.۱: قابلیت حمل کد جاوا

[۱] Java Virtual Machine

۴.۴.۱ سیستم عامل

یک سیستم کامپیوتری پیشرفته از یک یا چند پردازنده ، مقداری حافظه اصلی ، دیسک‌ها ، چاپگرها، صفحه کلید ، صفحه نمایش ، واسط‌های شبکه‌ای و دیگر دستگاه‌های ورودی و خروجی تشکیل شده است. این اجزا در کنار یکدیگر یک سیستم پیچیده را به وجود آورده اند. نوشتن برنامه‌هایی که تمامی این عناصر را مدیریت کرده و از آن‌ها به طور صحیح، بهینه و کارآمد استفاده نماید، کار بسیار مشکلی است. به همین دلیل، لایه نرم‌افزاری روی سخت‌افزار ایجاد شد که همه اجزای سیستم را کنترل نموده و کار برنامه نویسان را راحت‌تر کند. به این لایه نرم‌افزاری سیستم عامل می‌گویند. سیستم عامل بستری را فراهم می‌سازد که برنامه‌های کاربردی می‌توانند بر روی آن اجرا شوند. از آنجا که سیستم عامل وظیفه کنترل ارتباط بین محیط خارج و برنامه‌های کاربردی را دارد، دانش کلی راجع به نحوه کار آن برای انجام مهندسی معکوس لازم است.

۵.۱ شروع عملیات مهندسی معکوس

روش‌های مختلفی برای انجام مهندسی معکوس وجود دارد که ساده‌ترین آن‌ها این فرایند را به دو بخش تقسیم می‌کند. بخش اول معکوس‌سازی در سطح سیستم[۱] است و هدف از انجام آن درک ساختار کلی برنامه و همچنین پیدا کردن بخش‌های پراهمیت برای معکوس سازی می‌باشد. بخش دوم یا معکوس‌سازی در سطح کد[۲] به بررسی عمیق و به دست آوردن اطلاعات کامل قسمتی از کد برنامه می‌پردازد. در ادامه این دو مرحله و ابزار مورد نیاز برای انجام آن‌ها توضیح داده شده است.

[۱] System-Level Reversing

[۲] Code-Level Reversing

۱.۵.۱ معکوس‌سازی در سطح سیستم

این مرحله شامل اجرای ابزارهای گوناگون روی برنامه و استفاده از خدمات مختلف سیستم عامل به منظور به دست آوردن اطلاعات، بررسی برنامه‌ها و پیدا کردن ورودی و خروجی آن‌ها است. بیشتر این اطلاعات از طریق سیستم عامل به دست می‌آید. زیرا بنابر تعریف هر تعاملی که برنامه با محیط خارج دارد، از طریق سیستم عامل انجام می‌شود. بنابراین تسلط بر سیستم عامل در حین معکوس‌سازی باعث به دست آوردن اطلاعات ارزشمند در مورد برنامه مورد بررسی می‌شود. مقدمات سیستم عامل در فصل ۳ و توضیحات جامع برای معکوس‌سازی در سطح سیستم در فصل ۴ بررسی خواهند شد.

۲.۵.۱ معکوس‌سازی در سطح کد

استخراج مفاهیم طراحی و الگوریتم‌ها از برنامه دودویی فرایند پیچیده‌ای است که نیازمند تسلط به روش‌های معکوس‌سازی در کنار دانش تولید نرم‌افزار، پردازنده و سیستم عامل می‌باشد. نرم‌افزار ممکن است به قدری پیچیده باشد که حتی در صورت وجود مستندات مناسب و کافی، درک آن برای برنامه‌نویسان دشوار باشد. در این مرحله از معکوس‌سازی، کد از سطح بسیار پایین بررسی می‌شود و همه جزئیات نحوه عملکرد نرم‌افزار مشاهده می‌شود. اکثر این جزئیات به صورت خودکار توسط مترجم تولید شده‌اند و تولیدکننده نرم‌افزار در ایجاد آن‌ها نقشی نداشته است. همین امر باعث دشوار شدن فهم ارتباط بین کد و عملکرد نرم‌افزار می‌شود. بنابراین پیش از بررسی روش‌های واقعی معکوس‌سازی در سطح کد، درک نحوه عملکرد مترجم‌ها، زبان اسمبلی و ارتباط بین برنامه‌نویسی سطح بالا و سطح پایین نیاز است.

۶.۱ ابزارها

بخش زیادی از عملیات معکوس‌سازی توسط ابزارها انجام می‌شود. ابزارهایی که در زیر آمده‌اند به منظور انجام مهندسی معکوس طراحی نشده‌اند اما به انجام مرحله معکوس‌سازی در سطح کد کمک می‌کنند:

۱.۶.۱ ابزارهای پایش بر سیستم

برای بخش معکوس‌سازی در سطح سیستم، ابزارهایی نیاز است که اطلاعات کلی در مورد فایل اجرایی برنامه را در اختیار قرار دهند. معمولاً این اطلاعات توسط سیستم عامل ایجاد می‌گردد. براي مثال مشخص می‌گردد که نرم‌افزار با چه ابزاري توسعه داده شده و یا با چه ابزاری امنیت آن حفظ می‌شود. این ابزارها همچنین پایش فعالیت شبکه، دسترسی فایل‌ها، دسترسی به بانک اطلاعاتی و ...را بر عهده دارند. همچنین ابزارهایی برای نشان دادن میزان استفاده برنامه از اجزای سیستم عامل وجود دارند که در فصل ۴ بررسی خواهند شد.

۲.۶.۱ دیس‌اسمبلرها

همانطور که پیش‌تر گفته شد، دیس‌اسمبلر ابزاری است که یک فایل اجرایی را دریافت نموده و یک فایل متنی شامل کدهای زبان اسمبلی که مربوط به کل برنامه یا قسمت‌هایی از آن است را تولید می‌کند. از آنجا که زبان اسمبلی نمایش متنی کد زبان ماشین است، این فرایند نسبتاً ساده می‌باشد. دیس‌اسمبلر های پیشرفته یکی از ابزارهای کلیدی مهندسی معکوس به شمار می‌روند.

۳.۶.۱ اشکال‌زداها

اشکال‌زدا برنامه‌ای است که به توسعه دهنده این امکان را می‌دهد که برنامه را در حال اجرا مشاهده نماید. یکی از ویژگی‌های اساسی هر اشکال‌زدا توانایی قرار دادن نقاط انفصال[۱] است. نقاط انفصال نرم‌افزاری دستوراتی هستند که در زمان اجرای برنامه توسط اشکال‌زدا به برنامه اضافه شده و باعث می‌شوند کنترل برنامه به اشکال‌زدا سپرده شود. در این صورت هنگامی که برنامه به نقطه انفصال برسد، متوقف شده و وضعیت فعلی نمایش داده می‌شود. ویژگی دیگر اشکال‌زدا قابلیت دنبال کردن برنامه در حین اجرا است. برنامه پس از اجرای هر خط کد متوقف شده و به برنامه‌نویس امکان مشاهده و یا تغییر وضعیت فعلی را می‌دهد و به این ترتیب محتوای فایل اجرایی قابل تجزیه و تحلیل می‌شود. اشکال‌زداها از یک دیس‌اسمبلر برای تبدیل کدها به زبان اسمبلی استفاده می‌کنند و همچنین این امکان را فراهم می‌کنند که معکوسکننده رفتار پردازنده را با اجرای هر دستورالعمل بررسی کند. در بسیاری از مواقع یک اشکال‌زدا با دیس‌اسمبلر قوی برای انجام موفقیت‌آمیز مهندسی معکوس نقش کلیدی دارد.

۴.۶.۱ دی‌کامپایلرها

دی‌کامپایلر یا مترجم وارون برنامه‌ای است که کد زبان اسمبلی یا کد ماشین را به کد سطح بالای آن برمی‌گرداند این کار بسیار مشکل است زیرا نوشتن کدهای اسمبلی که منبع سطح بالایی برای آن‌ها وجود ندارد، امکان پذیر نمی‌باشد.

[۱] Breakpoints

فصل ۲

نرم‌افزارهای سطح پایین

پیچیدگی مهندسی معکوس زمانی به وجود می‌آید که تلاش برای پیدا کردن ارتباط بین نرم‌افزارهای سطح بالا و مفاهیم سطح پایین شروع می‌شود. در این فصل ابتدا نحوه نمایش ساختارهای ساده مانند ساختمان داده‌ها و جریان کنترل در سطح پایین بررسی می‌شود. سپس زبان اسمبلی برای معماری IA-۳۲ Intel بررسی شده و ثبات‌های آن معرفی می‌شوند.

۱.۲ مدیریت داده در سطح پایین

یکی از مهم‌ترین تفاوت‌های زبان‌های برنامه‌نویسی سطح بالا با نمایش‌های مختلف برنامه در سطح پایین در مدیریت داده است. زبان‌های برنامه‌نویسی سطح بالا، حتی زبان‌هایی مانند C ANSI که به زبان‌های سطح پایین نزدیک هستند، بخش زیادی از جزئیات مدیریت داده را از تولیدکننده پنهان می‌کنند. برای مثال قطعه کد ساده زیر که به زبان C است را در نظر بگیرید:

```
int Multiply(int x, int y)
{
int z;
z = x * y;
return z;
}
```

این تابع با وجود این که ساده به نظر می‌رسد، نمی‌تواند مستقیماً به معادل سطح پایین ترجمه شود. پردازنده‌ها به ندرت دستورالعملی برای شناختن یک متغیر یا ضرب دو متغیر و ذخیره حاصل در سومی دارند. محدودیت‌های سخت‌افزاری و ملاحظات زمان اجرا سطح پیچیدگی دستورالعمل‌ها را محدود می‌کند و حتی قدرتمندترین دستورالعمل‌هایی که پردازنده‌های Intel IA-۳۲ پشتیبانی می‌کند، در مقایسه با زبان‌های سطح بالا ابتدایی و ساده هستند.

بنابراین نمایش سطح پایین تابع Mutliply شامل مراحل زیر می‌شود:

۱. پیش از اجرای تابع وضعیت ماشین را ذخیره کن.

۲. حافظه‌ای به z اختصاص بده.

۳. x و y را از ثبات بیاور.

۴. x و y را ضرب کرده و حاصل را در یک ثبات ذخیره کن.

۵. نتیجه ضرب را در حافظه تخصیص داده شده به z بنویس.

۶. حالت ماشین ذخیره شده در مرحله ۱ را بازیابی کن.

۷. مقدار z را به عنوان خروجی برگردان.

مشاهده می‌شود که بیشتر پیچیدگی به وجود آمده به دلیل ملاحظات مدیریت داده است. در ادامه ساختارهایی نظیر ثبات‌ها، انباره‌ها و هرم‌ها و ارتباط آن‌ها با مفاهیم سطح بالا مانند متغیرها و پارامترها بررسی می‌شود.

۱.۱.۲ ثبات‌ها

برای جلوگیری از دسترسی به حافظه اصلی برای هر دستورالعمل، ریزپردازنده‌ها از حافظه‌ای داخلی استفاده می‌کنند که به راحتی قابل دسترسی است. ثبات‌ها قسمتی از حافظه موقت ریزپردازنده هستند که برای ذخیره و انتقال داده‌ها و دستورالعمل‌ها با سرعت بسیار بالا مورد استفاده قرار می‌گیرند. مشکل ثبات‌ها تعداد بسیار کم آن‌ها است. به عنوان مثال پردازنده Intel IA-۳۲ تنها ۸ ثبات ۳۲ بیتی دارد. البته تعداد دیگری ثبات وجود دارند اما از آن‌جا که برای مقاصد خاصی به کار گرفته شده‌اند، همیشه قابل استفاده نیستند. ثبات‌ها در بیشتر مواقع برای مدیریت و دسترسی داده‌های فوری استفاده می‌شوند و برای داده‌های دیگر از حافظه اصلی استفاده می‌شود. مدیریت داده توسط پردازنده به صورت خودکار انجام نمی‌شود. در نتیجه این وظیفه بر عهده زبان اسمبلی است و با توجه مجدد به تابع Multiply می‌توان نتیجه گرفت که پیچیدگی به وجود آمده به دلیل مدیریت داده و ذخیره اطلاعات از ثبات به حافظه اصلی و برعکس است.

۲.۱.۲ انباره‌ها

انباره قسمتی از حافظه برنامه است که برای ذخیره‌سازی کوتاه مدت داده‌ها توسط پردازنده مورد استفاده قرار می‌گیرد. ثبات‌ها برای ذخیره داده‌های فوری استفاده شده و پس از آن از انباره‌ها برای

نگهداری داده‌های کمی بلند مدت‌تر استفاده می‌شود. سیستم عامل‌های مدرن در لحظه چندین انباره مختلف را برای برنامه‌های در حال اجرا مدیریت می‌کنند. این ساختمان‌داده یک لیست خطی است که عملیات حذف و اضافه تنها از یك طرف آن به نام Top صورت می‌گیرد. در هر لحظه فقط عنصر بالایی انباره قابل دسترس است و آخرین عنصری که به انباره اضافه می شود اولین عنصری است که از آن حذف می شود به همین دلیل می‌توان انباره را ساختمان داده به عکس ترتیب ورود هم نامید. عمل اضافه کردن عنصر جدیدPUSH و عمل حذف عنصرPOP نامیده می‌شود. عملیات دیگری نظیر خواندن عنصر بالای انباره یا بررسی خالی بودن آن نیز ممکن است در برنامه‌های مختلف به کار بیاید. اگر بخواهیم کارایی انباره را در سطح بالا ترجمه کنیم، مشاهده می‌شود که هنگام فراخوانی و اجرای تابع، پارامترها و همه متغیرهای داخلی تابع در حافظه انباره قرار می‌گیرند. ذخیره متغیرهای محلی که در ثبات‌ها قرار نگرفته‌اند نیز توسط انباره انجام می‌شود.

۳.۱.۲ هرم‌ها

حافظه هرم بخشی مدیریت شده از حافظه است که اجازه‌ی اختصاص حافظه پویا به متغیرهای با اندازه متغیر در زمان اجرا را می‌دهد. تخصیص حافظه به این صورت انجام می‌شود که برنامه درخواست یک بلوک با اندازه مشخص می‌کند و یک اشاره‌گر در صورت وجود حافظه کافی به آن تعلق می‌گیرد. هرم‌ها به طور معمول توسط کتابخانه‌های نرم‌افزاری و یا سیستم عامل مدیریت می‌شوند. در مهندسی معکوس، درست قرار دادن هرم‌ها در حافظه و تشخیص مناسب تخصیص حافظه از هرم مهم است، زیرا این امر به درک کلی طرح برنامه کمک می‌کند.

۴.۱.۲ بخش‌های قابل اجرای داده‌ها

قسمت دیگری از حافظه برنامه که برای ذخیره داده کاربردی استفاده می‌شود، بخش قابل اجرای داده است. در زبان‌های سطح بالا این بخش شامل متغیرهای سراسری یا داده‌های از پیش مقداردهی شده(شامل ثابت‌ها و کدهای اختصاصی) است. معمولاً این داده‌ها مانند مقادیر ثابت صحیح داخل کد برنامه قرار می‌گیرند. اما زمانی که حجم داده زیاد باشد، مترجم آن را داخل بخش مخصوصی در فایل اجرایی برنامه ذخیره کرده و کدی تولید می‌کند که با آدرس به آن ارجاع داده شود. مثالی از داده‌های از پیش مقدار داده شده رشته‌های اختصاصی داخل یک برنامه هستند. به عنوان مثال قطعه کد زیر به زبان C را در نظر بگیرید:

```
char szWelcome = "This string will be stored in the executable's
preinitialized data section";
```

این تعریف، بدون توجه به این که کد szWelcome کجا تعریف شده، آن را در بخش قابل اجرای داده‌های از پیش مقداردهی شده قرار می‌دهد. برای دسترسی به این رشته، مترجم آدرسی اختصاصی که به آن اشاره می‌کند را تولید می‌کند. از آنجا که این آدرس‌ها به ندرت برای هدفی غیر از اشاره به بخش قابل اجرای داده استفاده می‌شوند، تشخیص آن‌ها در زمان معکوس کردن راحت است.

یکی دیگر از مواقعی که برای ذخیره از بخش قابل اجرای داده استفاده می‌کنیم، متغیرهای سراسری هستند. مقادیر این متغیرها تا پایان عمر برنامه باقی می‌ماند و ذخیره بلند مدت دارند. همانند داده‌های از پیش مقدار داده شده، متغیرهای سراسری هم توسط آدرس‌های مخصوص استفاده می‌شوند و بنابراین هنگام معکوس کردن به آسانی قابل تشخیص هستند.

۲.۲ جریان کنترل

دستورات جریان کنترل، جریان برنامه را تحت تاثیر قرار می‌دهند. در زبان‌های سطح بالا این دستورها به صورت حلقه‌های ساده شرطی هستند که توسط مترجم به دستورهای جریان کنترل سطح پایین تبدیل می‌شوند. از ساختارهای جریان کنترل می‌توان به بلوک‌های شرطی، حلقه‌ها و دستور switch اشاره کرد.

از آنجا که دستورات جریان کنترل در سطح پایین تقریباً ابتدایی هستند، نگاه کردن به پیاده‌سازی دستورات سطح بالا در سطح پایین پیچیده است. بنابراین روشی لازم است که مجدداً این دستورات ابتدایی را به مفاهیم کاربرپسند سطح بالا تبدیل کند. یکی از مشکلات این کار، بلند بودن دستورات شرطی سطح بالا برای تبدیل به زبان اسمبلی است. از طرفی دستورات زبان اسمبلی مستقل از محیط نیستند. در ادامه ساختار دستورهای جریان کنترل در زبان اسمبلی بررسی می‌شود.

۳.۲ زبان اسمبلی

یک برنامه اسمبلی مانند برنامه‌های سطح بالا به صورت متنی نوشته می‌شود. هر دستورالعمل زبان اسمبلی یک کد الفبایی از یک دستورالعمل ماشین است، که به این صورت معنی دستور واضح‌تر از کد زبان ماشین می‌شود. بین عبارات زبان اسمبلی و دستورالعمل‌های زبان ماشین تناظر یک به یک برقرار است. یعنی هر دستورالعمل اسمبلی دقیقاً یک دستورالعمل زبان ماشین را نشان می‌دهد و بالعکس، در حالی که در زبان سطح بالا یک عبارت معمولا به چندین دستورالعمل ماشین تبدیل می‌شود.

۱.۳.۲ ثبات‌ها

برای شروع یادگیری زبان اسمبلی ابتدا ۸ ثبات عمومی معماری IA-۳۲ لازم است معرفی شوند. نام تمام این ثبات‌ها با حرف E شروع شده‌اند که مخفف واژه Extended به معنای گسترش یافته است. نام‌گذاری این ثبات‌ها از روی معماری قدیمی‌تر ۱۶ بیتی Intel گرفته شده و گاهی اوقات در کدهای ۳۲ بیتی ممکن است نام قدیمی این ثبات‌ها مورد استفاده قرار گیرد. در زیر و شکل۱.۲ توضیح مختصری راجع به این ثبات‌ها و کاربرد آن‌ها بیان می‌کند.

EAX:این ثبات در عملیات ورودی خروجی و محاسبات زیاد استفاده می‌شود. ثبات AX به دو بخش AL و AH تقسیم می شود که هر کدام ۸ بیت فضا دارند.

EBX : این ثبات به عنوان شاخص برای توسعه آدرس استفاده می‌شود و به ثبات پایه معروف است. این ثبات در محاسبات نیز به کار می‌رود. ثبات BX نیز به دو بخش BL و BH تقسیم می‌شود که هر کدام ۸ بیت فضا دارند.

ECX: به ثبات شمارنده معروف است، و برای کنترل حلقه تکرار مورد استفاده قرار می گیرد. در عملیات جابه‌جایی، می توان تعداد جابه‌جاییها را در آن قرار داد. این ثبات در انجام محاسبات نیز استفاده می‌شود و به ثبات‌های CL و CH تقسیم شده است و هر کدام ۸ بیت فضا دارند.

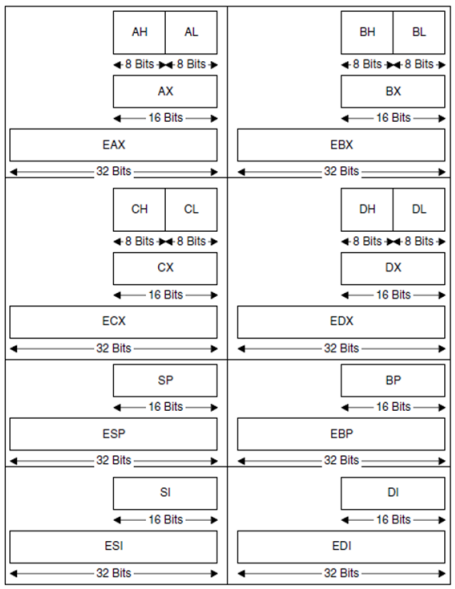
EDX: عملیات ضرب و تقسیم بزرگ با این ثبات انجام می‌شود. این ثبات دربعضی از اعمال ورودی و خروجی نیز به کار می‌رود و به ثبات داده‌ها معروف است. ثبات DX نیز به دو ثبات DL و DH تقسیم می‌شود که هر کدام ۸ بیت فضا دارند.

$ESI(SourceIndex)$: در عملیات رشته‌ای آدرس رشته منبع را در خود نگه می‌دارد.

$EDI(Destinationindex)$: در عملیات رشته‌ای آدرس رشته مقصد را در خود نگه می‌دارد.

$(BasePointer)EBP$:به عنوان اشاره‌گر به انباره شناخته شده و در فراخوانی زیر برنامه‌ها مورد استفاده قرار می گیرد.

$(StackPointer)ESP$: این ثبات آدرس فعلی بالای انباره را در خود نگه می‌دارد.



شکل ۱.۲: ثبات‌های پرکاربرد زبان اسمبلی

۲.۳.۲ دستورها

یک برنامه پیچیده از کنار هم قرار دادن دستورات ساده اسمبلی شکل می‌گیرد. هنگام شروع برنامه نویسی به زبان اسمبلی نیاز به یادگیری پرکاربردترین دستورات IA-۳۲ است که در ادامه گروه‌بندی شده و شرح داده خواهند شد.

دستورهای جابه‌جایی و انتقال داده

ساده ترین و احتمالا پرکاربردترین دستورالعمل در زبان اسمبلی MOV است که دارای دو عملوند می‌باشد . این دستورالعمل محتوای عملوند دوم(منبع) خود را به عملوند اول(مقصد) انتقال می‌دهد. عملوند مقصد می‌تواند یک آدرس حافظه و یا یک ثبات باشد و عملوند منبع هم می‌تواند داده فوری، آدرس حافظه و یا ثبات باشد. باید توجه داشت که انتقال حافظه به حافظه وجود ندارد. یعنی هردو عملوند همزمان نمی‌توانند عملوند حافظه‌ای باشند.

```
MOV  DestinationOperand, SourceOperand
```

دستورهای مقایسه‌ای و ریاضیاتی

برای مقایسه دو عملوند از دستورالعمل CMP استفاده می‌شود. دستور زیر را درنظر بگیرید:

```
cmp AX, BX
```

این دستور حاصل $AX - BX$ را محاسبه کرده و با توجه به حاصل نشانه‌ها را به صورت زیر تغییر می‌دهد. این نشانه‌ها برای بررسی نتیجه مقایسه بکار برده می‌شوند: نشانه Zero یک می‌شود اگر $AX = BX$ باشد و مساوی یا نامساوی بودن دو عملوند را مشخص می‌کند.
نشانه Carry وقتی یک می‌شود که در محاسبات $AX < BX$ بدون علامت بوده و رقم قرضی داشته باشد.
نشانه Sign به همراه نشانه Overflow در محاسبات علامتدار نشان می‌دهد کدام عملوند بزرگتر است.
دستورالعمل cmp روی نشانه های Parity و Carry Auxiliary هم تاثیر دارد ولی به ندرت

هنگام مقایسه مورد بررسی قرار می‌گیرند.

در زیر دستور ریاضی و شکل کلی آن‌ها در IA-۳۲ آمده است.

ADD، Operand Operand: حاصل جمع صحیح دو عملوند خود را محاسبه و نتیجه را در عملوند اول قرار می دهد.

SUB، Operand Operand: حاصل تفریق عملوند دوم از عملوند اول را محاسبه کرده و نتیجه را در عملوند اول قرار می‌دهد.

MUL Operand:عمل ضرب بدون علامت را انجام می‌دهد. این دستور دارای یک عملوند است و عملوند دیگر همیشه ثبات انباشتگر (AL/AX) درنظر گرفته می‌شود.

DIV Operand:تقسیم اعداد بدون علامت را انجام می‌دهد. عملوند دستورالعمل div مقسوم علیه تقسیم است و مقسوم بستگی به اندازه عملوند یکی از دو حالت زیر می‌باشد:

اگر عملوند ۸ بیتی باشد ثبات AX بر عملوند تقسیم می‌شود. خارج قسمت تقسیم در AL و باقیمانده در AH قرار می گیرند.

اگر عملوند ۱۶بیتی باشد مقدار ۳۲بیتی DX:AX بر عملوند تقسیم می‌شود. خارج قسمت تقسیم در AX و باقیمانده در DX ذخیره می‌شوند.

IMUL Operand: مشابه دستور mul است اما روی عملوندهای علامتدار عمل کرده و علامت حاصل ضرب را با توجه به بیت علامت عملوندها تنظیم می‌کند.

IMUL Operand: مشابه دستورالعمل div است با این تفاوت که تقسیم علامتدار را انجام می دهد.

دستورهای جریان کنترل برنامه

پردازنده دستورها را به ترتیبی که در برنامه ظاهر شده‌اند اجرا می‌کند. ساختارهای کنترلی نظیر عبارات شرطی، حلقه‌ها و فراخوانی زیربرنامه روال اجرای برنامه را تغییر می‌دهند. زبان‌های سطح بالا ساختارهای کنترلی سطح بالا مانند دستورات if و while را دراختیار می‌گذارند که اجرای برنامه را کنترل می‌کنند. زبان اسمبلی از دستورات پرش برای پیاده‌سازی این ساختارهای کنترلی استفاده می‌کند. شکل عمومی این دستورها به صورت زیر است.

```
Jcc TargetCodeAddress
```

در صورت برقراری شرط مورد نظر، Jcc اشاره‌گر را به‌روزرسانی می‌کند تا به TargetCodeAddress اشاره کند. در غیر این صورت بدون توجه به دستورهای داخل Jcc، ادامه دستورها اجرا می‌شود.

دستورالعمل jmp مشابه دستور goto در زبان‌های سطح بالا عمل کرده و بدون هیچ شرطی کنترل را به نقطه دیگری در برنامه منتقل می‌کند و شکل کلی آن به صورت زیر است.

```
jmp target
```

دستورهای فراخوانی توابع

فراخوانی توابع در زبان اسمبلی با دستور CALL و برگرداندن مقدار با دستور RET پیاده‌سازی می‌شود. دستور CALL دستورالعمل فعلی را داخل انباره PUSH می‌کند و امکان برگرداندن مقدار را از این طریق فراهم می‌سازد. سپس به آدرس مشخصی پرش می‌کند. آدرس تابع مانند هر عملوند دیگری می‌تواند مشخص شود. شکل کلی دستور CALL به صورت زیر است.

```
CALL FunctionAddress
```

پس از اتمام عملیات تابع، دستور RET برای برگرداندن جواب فراخوانی می‌شود. این دستور اشاره‌گری که توسط CALL به داخل انباره PUSH شده بود، POP می‌کند.

دستورهای معرفی شده در این فصل تنها بخش کوچک اما پرکاربردی از دستورات زبان اسمبلی هستند. برای کاستن از پیچیدگی‌های این زبان و برنامه‌نویسی سطح پایین و همچنین تسریع فرایند معکوس‌سازی کد، ابزارهایی وجود دارد که در فصل ۳ به تفصیل آمده‌اند.

فصل ۳

ابزارهای معکوس‌سازی

بدون ابزار مناسب معکوس کردن کد ممکن نیست. صدها نرم‌افزار به منظور ممکن کردن معکوس‌سازی وجود دارد که بعضی از آن‌ها رایگان و بعضی دیگر بسیار هزینه‌بر است. ولی آنچه مهم است، شناخت تفاوت این ابزارها و انتخاب نرم‌افزار مناسب است. در این بخش ابزارهای موجود مختلف معرفی شده و بهترین کاربردهای آن‌ها بررسی می‌شود.

۱.۳ روش‌های معکوس‌سازی

روش‌های مختلفی برای معکوس کردن کد موجود است و انتخاب این روش بستگی به برنامه هدف، پلتفرمی که کد روی آن نوشته شده، پلتفرمی که کد روی آن اجرا می‌شود و نوع اطلاعات مورد نیاز برای استخراج دارد. به طور کلی دو روش اساسی تجزیه و تحلیل برون‌خط[۲] و تجزیه و تحلیل برخط

[۲] Offline Code Analysis (Dead-Listing)

[1] وجود دارد.

۱.۱.۳ تجزیه و تحلیل برون‌خط

تحلیل برون‌خط کد به معنای تبدیل کد دودویی اجرایی به برنامه قابل خواندن توسط انسان به وسیلهی ابزارهای دیس‌اسمبلر و دی‌کامپایلر می‌باشد. سپس معکوس کردن با خواندن و تحلیل بخش‌های مختلف خروجی انجام می‌شود.

تحلیل برون‌خط رویکردی قدرتمند است زیرا طرح کلی مناسبی از برنامه ارائه داده و جستجوی توابع مورد علاقه را آسان می‌کند.

یکی از مشکلات رویکرد برون‌خط این است که در مقایسه با تجزیه و تحلیل برخط، نیاز به درک بهتری از کد دارد زیرا جریان برنامه و اطلاعات قابل مشاهده نیستند و گاهی مجبور به حدس زدن این اطلاعات می‌شویم. علاوه بر پیچیدگی، در برخی موارد این روش کاربردی ندارد. به عنوان مثال برای برنامه‌های بسته که کد آن‌ها تا زمان اجرا فشرده‌سازی و رمزنگاری شده، تنها از تجزیه و تحلیل برخط می‌توان استفاده کرد.

۲.۱.۳ تجزیه و تحلیل برخط

تجزیه و تحلیل برخط همان مراحل رویکرد اول را برای تبدیل کد در بر می‌گیرد، با این تفاوت که این بار کد معکوس شده در یک اشکال‌زدا اجرا شده و رفتار آن مورد بررسی قرار می‌گیرد. به دلیل مشاهده داده‌های درونی و تاثیر آن‌ها بر جریان کد، این رویکرد اطلاعات بیشتری فراهم می‌کند. همچنین مقادیر متغیرها قبل و بعد از اجرای برنامه قابل مشاهده است. برای شروع عملیات معکوس

[1] Live Code Analysis

کردن، استفاده از تجزیه و تحلیل برخط به علت اطلاعات مفیدی که فراهم می‌کند، توصیه شده است. در ادامه ابزارهای قابل استفاده این روش بررسی می‌شوند.

۳.۱.۳ اشکال‌زدا

کاربرد اول اشکال‌زدا کمک به تولیدکننده برای نشان دادن و اصلاح اشتباهات موجود در کد است اما از آن می‌توان به عنوان یک ابزار قدرتمند مهندسی معکوس نیز استفاده کرد. برنامه‌های اشکال‌زدا که دارای خواندن کدهای زبان اسمبلی بدون کد منبع هستند، بهترین ابزار برای معکوس‌سازی محسوب می‌شوند. ایده استفاده از اشکال‌زدا این است که دید دقیق به تابع در حال اجرا داده شود و نشان دهد که در هر خط این برنامه‌ی دیس‌اسمبل شده چه اتفاقی می‌افتد. در حین پیشروی در کد، وضعیت ثبات‌های پردازنده، حافظه مصرفی و انباره فعال نشان داده می‌شود.
در ادامه ویژگی‌های اصلی یک اشکال‌زدا برای مهندسی معکوس آمده است

۱. دیس‌اسمبلر قدرتمند برای مشاهده کامل کد و تشخیص جریان حرکت دستورالعمل‌های مختلف به منظور معکوس کردن راحت‌تر

۲. انفصال نرم‌افزاری[۱] به منظور متوقف کردن اجرای برنامه و منتقل کردن کنترل برنامه به اشکال‌زدا.

۳. انفصال انفصال سخت‌افزاری[۲] به منظور متوقف کردن اجرای برنامه توسط پردازنده زمانی که یک آدرس حافظه مشخص دستیابی می‌شود، و منتقل کردن کنترل برنامه به اشکال‌زدا

[۱] Software breakpoints

[۲] Hardware breakpoints

۴. دید کلی و مناسب به ثبات‌های پردازنده و حافظه سیستم.

۵. اطلاعات پردازش دقیق در حین اشکال‌زدایی، مانند اطلاعات راجع به فهرست ماژول‌های اجرایی و فهرست دستورالعمل‌های در حال اجرا

در سیستم عامل‌های مدرن، اشکال‌زداها به دو نوع تقسیم می‌شوند. دسته اول اشکال‌زداهای user-mode هستند که معمولا توسط تولیدکنندگان نرم‌افزار استفاده می‌شوند و همان طور که نام‌گذاری آن‌ها نشان می‌دهد، مانند سایر برنامه‌ها در حالت کاربری اجرا می‌شوند. دسته دوم اشکال‌زداهای kernel-mode هستند که در مقایسه با اشکال‌زدای user-mode قدرت بسیار بیشتری دارند. آن‌ها امکان کنترل کامل سیستم هدف و مشاهده‌ی هر آن چه در سیستم رخ می‌دهد، صرف نظر از اینکه در داخل کد سیستم اتفاق افتاده است یا برنامه کاربردی، فراهم می‌کنند.
در ادامه مزایا و معایب این دو دسته بررسی شده و پرکاربردترین ابزارهای هر یک معرفی می‌شوند.

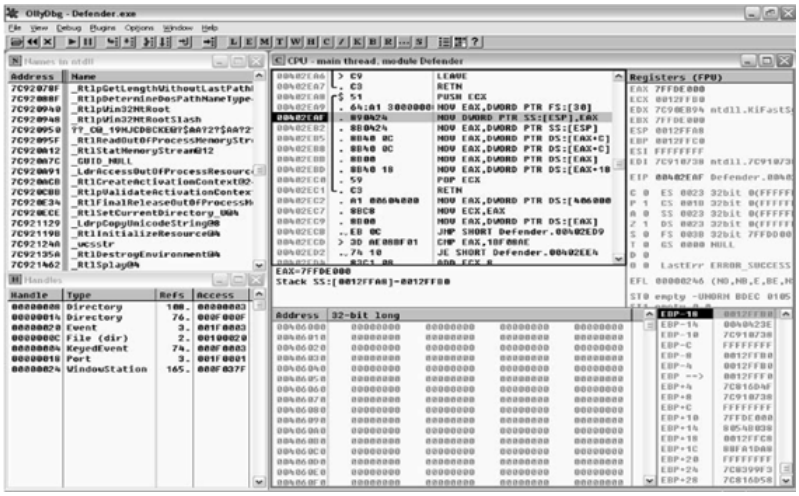
۴.۱.۳ اشکال‌زدای user-mode

این دسته از برنامه‌های اشکال‌زدا به دلیل ساده بودن استفاده و راه‌اندازی معروف هستند. اشکال‌زدای user-mode مانند برنامه‌ای کاربردی به فرایند دیگری که قصد اشکال‌زدایی آن را داریم متصل شده و کنترل آن را در دست می‌گیرد. مهم‌ترین مشکل این اشکال‌زدا این است که تنها توانایی مشاهده یک فرایند را دارد. به همین دلیل برای معکوس کردن باید فرایند مورد نظر را بدانیم. اطلاع از فرایند زمانی که برنامه شامل فرایندهای تو در تو و مرتبط به هم باشد بسیار دشوار است. محدود بودن به کد کاربر تا زمانی که برنامه شامل اجزایی از حالت kernel نباشد و نیازی به دسترسی به کد سیستم عامل نداشته باشیم، مشکلی ایجاد نمی‌کند. اما علاوه بر محدودیت گفته

شده، بعضی از اشکال‌زداهای user-mode قادر به اشکال‌زدایی برنامه قبل از رسیدن به فایل اجرایی آن نیستند. در ادامه ابزارهای اشکال‌زدای user-mode که در فرایند مهندسی معکوس نرم‌افزار کاربرد دارند آمده است.

OllyDbg

برای افرادی که به تازگی مهندسی معکوس را شروع کرده‌اند و یا مبتدیان، نرم‌افزار OllyDbg نوشته Yuschuk Oleh ابزاری مناسب است. این اشکال‌زدا به منظور انجام مهندسی معکوس طراحی شده و دارای یک دیس‌اسمبلر قدرتمند است. این دیس‌اسمبلر ویژگی تحلیل کد را فراهم کرده و حلقه‌های تکرار، عبارات شرطی، و پارامترهای مربوط به هر متد را به راحتی مشخص می‌کند. OllyDbg همچنین دارای خاصیت patching می‌باشد و به این ترتیب می‌توان در فایل اجرایی تغییراتی را ایجاد و ذخیره کرد. رایگان بودن OllyDbg باعث شده تا این نرم‌افزار در کسانی که نیازی به دستیابی به کد سیستم عامل و kernel ندارند، بسیار محبوب باشد. شکل ۱.۳ تصویری از محیط OllyDbg را نشان می‌دهد.

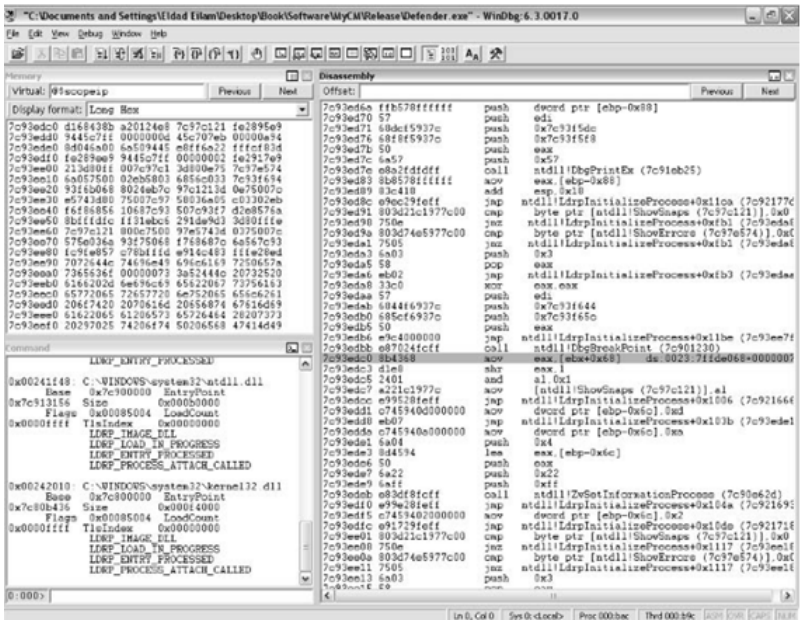


شکل ۱.۳ : محیط OllyDbg

OllyDbg

winDbg اشکال‌زدایی است که توسط مایکروسافت تولید شده و در قالب بسته ای به نام Debug- Windows for Tools ging از وبسایت مایکروسافت به صورت رایگان قابل دریافت است. این اشکال‌زدا از یک رابط خط فرمان به عنوان رابط کابری اصلی استفاده می‌کند و در نسخه‌های اخیر، برخی از ویژگی‌های آن توسط واسط گرافیکی کاربر نیز قابل دسترسی هستند. WinDbg محدودیت‌های آزاردهنده‌ای مانند ناتوانی در بازگشت به عقب در صفحه دارد که کار با آن را دشوار می‌کنند. اما از نظر یکپارچگی با سیستم‌عامل، به مراتب بهتر از OllyDbg عمل می‌کند. به علاوه بر خلاف OllyDbg که اشکال‌زدایی را در زمان شروع فایل اجرایی آغاز می‌کند، WinDbg این امکان را در مراحل اولیه مقداردهی فرایندها فراهم می‌کند. شکل ۲-۳ تصویری از محیط این نرم‌افزار را نشان می‌دهد.

در طی سال‌های اخیر، WinDbg پیشرفت چشمگیری داشته است به طوریکه نسخه‌های جدید و بهبود یافته با امکانات بیشتر به طور منظم منتشر می‌شوند. اما همچنان برای معکوس کردن برنامه‌هایی که یکپارچگی زیادی با سیستم عامل ندارند، استفاده از OllyBug به دلیل رابط کاربری بهتر، دیس‌اسمبلر قدرتمندتر و توانایی تجزیه و تحلیل کد توصیه می‌شود.



شکل ۲.۳: محیط WinDbg در حال اجرای فرایند

Pro IDA

Pro IDA دیس‌اسمبلر قدرتمند با قابلیت پشتیبانی از معماری‌های مختلف پردازنده مانند ،IA-۳۲ (Itanium)IA-۶۴ و AMD۶۴ می‌باشد. همچنین این دیس‌اسمبلر از فرمت‌های مختلف فایل مانند ،ELF PE و حتی XBE که بر روی کنسول بازی مایکروسافت Xbox اجرا می‌شود، پشتیبانی می‌کند. Pro IDA همچنین یک اشکال‌زدای user-mode قدرتمند است. دیس‌اسمبلر بسیار

قدرتمند و امکانات فراوان اشکال‌زدایی باعث شده که این نرم‌افزار با وجود قیمت بالای آن مورد استفاده قرار گیرد. IDA همچنین توانایی رسم فلوچارت و نمودار برای توابع مورد نظر را دارد. محیط IDA به دلیل امکانات زیاد کار را برای برنامه نویسان راحت کرده است به عنوان مثال، با کلیک کردن روی یک ثبات، تمام مواردی که به آن ارجاع داده شده‌اند نشان داده می‌شوند. این کار فرایند درک منطق کد و جریان داده را آسان و سریع می‌کند. Pro IDA به کسانی که قصد انجام مهندسی معکوس برای مدت طولانی را دارند، بهترین اشکال‌زدا و دیس‌اسمبلر موجود است.

۲.۳ اشکال‌زدای kernel-mode

این دسته از اشکال‌زداها به سیستم به عنوان یک موجودیت نگاه می‌کند و نه یک فرایند خاص. بر خلاف اشکال‌زدای user-mode، اشکال‌زدای kernel-mode برنامه‌ای که در سیستم عامل اجرا شود نیست و اجازه قطع و مشاهده کل سیستم را در هر زمان به برنامه‌نویس می‌دهد. تمامی وظایف اشکال‌زدای user-mode توسط این اشکال‌زدا قابل انجام است اما گاهی اوقات عدم آگاهی برنامه‌نویس به جابه‌جایی آدرس‌های حافظه در فرایندهای در حال اجرا ایجاد مشکل می‌کند. بنابراین با وجود توانایی بالا برای انجام اشکال‌زدایی در سطح کاربر، این اشکال‌زدا عموماً توسط معکوس‌کنندگان کد سیستم عامل استفاده می‌شود. به دلیل فراهم کردن امکان مشاهده کامل سیستم و فرایندهای در حال اجرا، بعضی از برنامه‌نویسان استفاده انحصاری اشکال‌زدای kernel-mode را ترجیح می‌دهند.

از دیگر کاربردهای اشکال‌زدای kernel-mode امکان قرار دادن نقاط انفصال در سطح پایین است. نقاط انفصال به تولیدکنندگان کمک می‌کنند تا در یک لحظه‌ی خاص یا هنگام رسیدن به یک

دستور معین، اطلاعات مورد نیاز نسبت به وضعیت برنامه را به دست آورد، مقدار متغیرها را بررسی کرده، در صورت لزوم تغییرات مورد نیاز را اعمال نموده و اجرای برنامه را از همان نقطه ادامه دهد. متاسفانه اشکال‌زداهای kernel-mode معمولاً راه‌اندازی سختی داشته و نیاز به سیستمی اختصاصی دارند. از طرفی از آنجا که تمام سیستم را کنترل می‌کنند، تا زمان استفاده از آن‌ها سیستم به طور موقت از کار می‌افتد. به دلیل محدودیت‌های گفته شده استفاده و نصب آن‌ها تنها برای زمانی که نیاز برنامه‌نویس با ابزارهای user-mode برطرف نشود توصیه می‌شود.

در ادامه ابزارهای اشکال‌زدای kernel-mode که در فرایند مهندسی معکوس نرم‌افزار کاربرد دارند آمده است.

WinDbg

در درجه اول،WinDbg یک اشکال‌زدای kernel-mode است. این اشکال‌زدا روی سیستمی جدا از سیستم اجراکننده واسط گرافیکی WinDbg اجرا می‌شود. سیستم هدف با استفاده از دستور DEBUG که در فایل پیکربندی boot.ini تنظیم می‌شود، راه اندازی شده و در داخل هسته سیستم عامل کد مخصوصی را برای اشکال‌زدایی فعال می‌کند. سیستم اصلی و سیستم هدف اشکال‌زدایی معمولاً توسط اتصال پرسرعت $(IEEE1394) FireWire$ با هم در ارتباط هستند.

در عمل WinDbg اشکال‌زدایی انعطاف‌پذیر و قدرتمند است، اما رابط کاربری نسبتاً ضعیفی دارد. به طوری که بسیاری از ویژگی‌های آن تنها از طریق $command - line$ قابل استفاده است. همچنین هنگام استفاده از WinDbg در حالت user-mode مشکلات و محدودیت‌هایی وجود دارد. یکی از این محدودیت‌ها که در نسخه‌های جدیدتر سیستم‌عامل تا حدودی برطرف شده، اشکال در قرار دادن نقاط انفصال توسط کاربر است. همچنین در صورت عدم استفاده از ماشین

مجازی و یا FireWire برای ایجاد اتصال دو سیستم، کارایی و سرعت اشکال‌زدایی به شدت کم می‌شود.

ماشین مجازی

همان طور که گفته شد، اشکال‌زدای kernel-mode سیستم‌عاملی که روی آن اجرا می‌شود را بی‌ثبات و متوقف می‌کند. بنابراین توصیه می‌شود از رایانه اصلی برای اجرای این اشکال‌زدا استفاده نشده و سیستمی جدا در نظر گرفته شود. اما تهیه یک سیستم اضافه هزینه‌بر بوده و همچنین جابه‌جا کردن آن ممکن نیست. راه حل این مشکل استفاده از ماشین مجازی است. ماشین مجازی یک همسان‌سازی از سیستم‌های رایانه‌ای است. به عبارت دیگر ماشین‌های مجازی می‌توانند رفتار و عملکرد یک رایانه فیزیکی را تقلید کنند. معمولا این همسان سازی حاصل ترکیبی از نرم‌افزار و سخت‌افزارهای ویژه می‌باشد. سیستم فیزیکی که ماشین مجازی روی آن اجرا می‌شود ماشین میزبان[۱] نامیده می‌شود و هر ماشین مجازی را نیز یک ماشین مهمان[۲] می‌نامند. استفاده از ماشین مجازی امکان اشکال‌زدایی همزمان(با فرض حافظه‌ی کافی میزبان) را بدون تاثیر بر ثبات ماشین میزبان فراهم می‌کند.

ماشین مجازی با در نظر گرفتن تمام سیستم روی یک فایل در ماشین میزبان کار مدیریت و پشتیبان‌گیری را برای کاربران راحت کرده است. به عنوان مثال می‌توان وضعیت فعلی سیستم را ذخیره کرد و سپس در تنظیمات آن تغییراتی را به وجود آورد. بازگشت به وضعیت قبلی سیستم بر خلاف سیستم‌های غیر مجازی تنها با کپی کردن فایل اصلی امکان‌پذیر است. علاوه بر این

[۱] Host machine

[۲] Guest machine

ماشین‌های مجازی از درایوهایی که با خاموش کردن و راه‌اندازی مجدد سیستم اطلاعات نوشته شده را از دست می‌دهند پشتیبانی می‌کند و این ویژگی برای تشخیص نرم‌افزارهای مخرب مناسب است.

متاسفانه ماشین مجازی منابع قابل توجهی را روی ماشین میزبان نیاز دارد. میزبان باید حافظه کافی برای سیستم عامل، برنامه‌های اجرا شده روی آن و فضای اختصاص یافته به برنامه‌های در حال اجرای ماشین مهمان داشته باشد. معمولا میزان حافظه اختصاص یافته به هر ماشین مهمان توسط کاربر قابل تغییر است. بعضی از ماشین‌های مجازی با توجه به پردازنده قابلیت تقلید دارند و می‌توانند هر سیستمی از هر پلتفرمی را تقلید کنند که این کار کارایی سیستم را پایین می‌آورد. کاربرد دیگر ماشین مجازی اجرای سیستم عامل مهمان که با پردازنده میزبان سازگار است می‌باشد. به این ترتیب اجازه اجرای مستقیم سیستم مهمان روی پردازنده میزبان تا حد امکان داده می‌شود. به نظر می‌رسد این تنها راه به دست آوردن کارایی مناسب از سیستم مهمان است، اما مشکل این است که اجرای مستقیم سیستم مهمان ممکن است با سیستم عامل میزبان تداخل داشته باشد. برای حل این مشکل ماشین‌های مجازی امروزی با بررسی کد مهمان از جدا بودن دو سیستم اطمینان پیدا می‌کنند.

در سال‌های اخیر فناوری ماشین‌های مجازی پیشرفت چشمگیری داشته و به عنوان راه‌حلی سریع و پایدار برای افرادی که نیاز به بیش از یک رایانه دارند عنوان می‌شود. در حال حاضر دو ماشین مجازی PC Virtual محصول شرکت مایکروسافت و Workstation VMWare محصول شرکت VMWare مورد استفاده قرار می‌گیرند. از نظر عملکرد این دو ماشین تقریبا مشابه هستند. هر دو امکان اجرا روی سیستم عامل‌های مختلف ویندوز و غیر ویندوز را دارند. همچنین هر دو از اشکال‌زدایی kernel-mode با ابزارهایی مانند WinDbg یا SoftICE NuMega به طور کامل

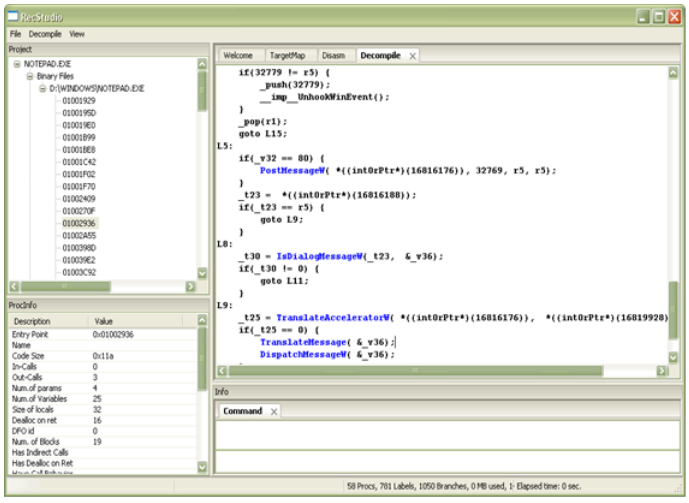
پشتیبانی می‌کنند. تنها تفاوت در این است که VMWare روی میزبان غیرویندوز مانند لینوکس (شکل ۳.۳) نیز اجرا شده و امکان اجرای ویندوز و یا نسخه های دیگر لینوکس را برای آن فراهم می‌کند.

شکل ۳.۳: محیط اجرای VMWare نسخه ۶ روی سیستم عامل لینوکس

نرم‌افزارهای مجازی‌ساز شرکت ،VMWare دو نسخه عمده دارند. نسخه رایانه‌های رومیزی یا دسکتاپ که با عنوان Workstation VMWare شناخته می‌شود و نسخه سرور آن که به Server ESXi VMWare معروف می‌باشد. ویژگی اصلی Server ESXi VMWare که آن را از نسخه Workstation VMWare متمایز می‌کند، این است که به صورت مستقیم بر روی سخت‌افزار نصب شده و دیگر نیازمند یک سیستم عامل به عنوان رابط نمی‌باشد، که در سرعت کارکرد سیستم بسیار مؤثر است.

۳.۳ دی‌کامپایلر

دی‌کامپایلرها برنامه‌هایی هستند که عمل معکوس کامپایلرها را انجام می‌دهند و کد ماشین یا کد اسمبلی میانی را به کد سطح بالا تبدیل می‌کنند و هدف انها ایجاد فایل قابل خواندن توسط کامپیوتر و در محله بعدی ترجمه این فایل به زبان قابل فهم توسط انسان است. البته از آنجاکه عمل کامپایل کردن همیشه مقداری از اطلاعات را حذف می‌کند، به دست آوردن خروجی از تمام قسمت‌های کد اصلی در عمل غیرممکن است. مقدار اطلاعات نگهداری شده در کد دودویی بستگی به کامپایلر استفاده شده، زبان سطح بالا و زبان سطح پایینی که برنامه توسط کامپایلر به آن ترجمه شده دارد. کد تولید شده در معماری IA-۳۲، حاوی اطلاعات سطح بالای کمی است و به همین دلیل بازیابی دقیق از نمایش سطح بالای آن هنوز امکان‌پذیر نیست. RECStudio یکی از قدرتمند ترین ابزارهای فعلی در مهندسی معکوس است که عمل دی‌کامپایل کردن در سطوح پایین را انجام می دهد. شکل ۴.۳ نمایی از محیط RECStudio را نشان می‌دهد.



شکل ۴.۳: محیط نرم‌افزار RECStudio

۴.۳ ابزارهای پایش سیستم

پایش سیستم بخش مهمی از فرایند معکوس‌سازی است. گاهی اوقات حتی بدون نگاه کردن به کد ممکن است فرد تنها با استفاده از ابزارهای پایش سیستم به پاسخ سوال خود برسد. این ابزارها راه‌های ارتباط ورودی و خروجی بین برنامه و سیستم‌عامل را پایش می‌کنند. به وسیله ابزارهای پایش سیستم همچنین می‌توان هر عملیات انجام شده روی فایل، مانند خواندن، نوشتن و ایجاد مطالب را مشاهده کرد. ابزارهای پایش مختلفی وجود دارند که درادامه خلاصه‌ای از آن‌ها آمده است.

FileMon : این ابزار وظیفه کنترل تمام جابه‌جایی‌های در سطح فایل و سیستم و بین برنامه‌ها و سیستم عامل را بر عهده دارد. همچنین برای مشاهده فایل ورودی/خروجی تولید شده توسط فرایندهای در حال اجرا در سیستم استفاده می‌شود. هرچند این ابزار دیگر توسط سیستم عامل‌های مایکروسافت پشتیبانی نمی‌شود، می‌توان آن را بخش مهمی از ابزار جدیدتر Monitor Process در نظر گرفت.

TCPView: این ابزار تمام شبکه‌های ارتباطی فعال قرارداد بسته داده کاربر[۱] و قرارداد کنترل انتقال[۲] را برای هر فرایند نشان می‌دهد. باید توجه داشت که TCPView مقدار واقعی جابه‌جا شده را نشان نمی‌دهد و تنها لیستی ار ارتباطات هر فرایند با نوع ارتباط آن را مشخص می‌کند.

Monitor Process: ابزاری رایگان از تولیدکننده نرم‌افزار Sysinternals که حاصل ترکیب دو ابزار FileMon و RegMon می‌باشد. همه‌ی فعالیت‌های فایل و سیستم توسط Process-Monitor به صورت بلادرنگ پایش می‌شوند. از دیگر کاربردهای آن می‌توان به اشکال‌زدایی نرم‌افزار، جرم‌یابی رایانه‌ای و مدیریت سیستم اشاره کرد.

[۱] User Datagram Protocol

[۲] Transmission Control Protocol

PortMon: ابزاری پیشرفته که برای پایش فعالیت پورت‌های سری یا موازی ورودی/خروجی استفاده می‌شود. اطلاعات گزارش شده توسط PortMon بر اساس فرایندها جدا می‌شوند.

WinObj : ابزاری برای مشاهده فضای نام‌ها است که نمایشی سلسله مراتبی از نام اشیا موجود در سیستم می‌دهد. این نحوه نمایش برای هماهنگ‌سازی نام‌ها و یا مشاهده و درک اشیا سیستم مفید است. مجموعه‌ای از بیش از۴۰ نرم‌افزار کم حجم ولی کاربردی برای پایش سیستم و ترمیم و مدیریت آن توسط توسعه دهنده Sysinternals و با نامSuite Sysinternals Windows به بازار عرضه شده است که نسخه ۲۰۱۸,۰۶,۰۱ آن شامل فایل اجرایی نرم‌افزارهای موجود در شکل ۵.۳می‌باشد

شکل ۵.۳: نرم‌افزارهای موجود در بسته نرم‌افزاری Suite Sysinternals Windows

استفاده از ابزارهای گفته شده در این فصل، برای هر معکوس‌کننده مبتدی الزامی است. در کنار این نرم‌افزارها، ابزارهای متفرقه برای به کار گیری و یا ایجاد وصله‌های نرم‌افزاری وجود دارد که در شرایط خاص مورد استفاده قرار می‌گیرند.

فصل ۴

بررسی قانونی

استفاده و تغییر غیرقانونی انواع مختلف محتوای دیجیتالی به خصوص نرم‌افزارها در سال‌های اخیر به مشکلی بزرگ تبدیل شده است. در این فصل به اختصار به بیان مزایا و معایب مهندسی معکوس و جنبه حقوقی و قانونی آن پرداخته و روش‌های جلوگیری از انجام مهندسی معکوس را بررسی می‌کنیم.

۱.۴ مزایا و معایب مهندسی معکوس نرم‌افزار

از نظر بسیاری از افراد، مهندسی معکوس تحولی است که لزوما باعث پیشرفت جوامع بشری نمی‌شود. این عمل ممکن است گاه باعث نقض حقوق مادی و معنوی پدیدآورندگان اصلی نرم‌افزار و استفاده از اثر دیگران بدون هیچ هزینه و تلاشی شود. در بسیاری از کشورهای در حال توسعه از جمله ایران شبیه‌سازی محصولات خارجی با باز کردن و کشف رموز آن‌ها رایج بوده و در عمل حمایت قانونی تنها بر محصولات داخلی اعمال می‌شود. اما برخلاف تصور افراد، شکستن قفل نرم‌افزار و استفاده

یا انتشار نامحدود آن هیچگاه در حوزه مهندسی معکوس قانونی نمی‌گنجد. باید توجه داشت که طبق تعریف مهندسی معکوس نرم‌افزار، همه مراحل تولید یک محصول نرم‌افزاری باید به صورت عکس انجام شود. در نتیجه شکستن قفل تنها بخشی از کل فرایند بوده و محصول باید با استفاده از طراحی مجدد بازیابی شده و یا از نو ساخته شود.

البته نباید از مزایای مهندسی معکوس به عنوان بهترین و گاه تنها روش ساخت مجموعه‌ای هدفمند از عناصری که به تنهایی ارزشی ندارند، چشم‌پوشی کرد. موارد زیر تنها بخشی از دلایل استفاده از مهندسی معکوس در تولید نرم‌افزار هستند.

۱. عدم وجود مستندات کافی از طراحی اصلی محصول.

۲. توقف تولید یک محصول توسط تولیدکننده اصلی.

۳. بهبود ویژگی‌های خوب و یا طراحی مجدد ویژگی‌های یک محصول.

۴. کشف راه‌های جدید به واسطه تجزیه و تحلیل محصولات رقیبان به منظور بهبود عملکرد محصول.

۵. به روز رسانی نرم‌افزارهای منسوخ و تولید محصول بهتر.

با توجه به آنچه گفته شد، پرسش اصلی این است که مهندسی معکوس تا چه حدی مجاز و قانونی است و تحت چه شرایطی می‌توان آن را ممنوع کرد؟

۲.۴ قوانین حمایتی و پیشگیری از مهندسی معکوس

برای جلوگیری از شبیه‌سازی نسخه اصلی نرم‌افزار توسط دیگران، ماده ۱۵ آیین‌نامه اجرایی مواد ۲ و ۱۷ قانون حمایت از حقوق پدیدآورندگان نرم‌افزارهای رایانهای، تصویب شده در تاریخ 1383/4/21 مقرر می‌دارد: "اشخاصی که نرم‌افزاری را با تغییراتی که عرفاً نتوان آن را یک نرم‌افزار جدید به حساب آورد، به نام خود ثبت، تکثیر، منتشر، عرضه و یا بهره‌برداری نمایند، حقوق پدیدآورنده نرم‌افزار یاد شده را نقض کرده‌اند".

برنامه‌نویسان باید توجه داشته باشند که با استفاده از مهندسی معکوس و ابزارهای معرفی شده در فصل گذشته، دسترسی به کد منبع از طریق فایل اجرایی امکان‌پذیر است. هر چند هنوز هیچ راهکار مطمئنی برای جلوگیری کامل از سوء استفاده و مهندسی معکوس غیرقانونی وجود ندارد، راهکارهای زیر این عمل را به تاخیر انداخته و فهم برنامه را دشوار می‌کند:

۱. استفاده از نرم‌افزارهای مخصوص برای هر زبان برنامه نویسی

به عنوان مثال نرم‌افزار RubyEncoder که محصول شرکت انگلیسی SourceGuardian است، کدهای زبان Ruby را به بایت کد تبدیل کرده و سپس آن‌ها را رمزنگاری می‌کند. این عمل باعث جلوگیری از مهندسی معکوس کد منبع می‌شود. Winlicense نرم‌افزار دیگری است که با استفاده از الگوریتم‌های مختلف رمزنگاری چند سطحی از کد و داده در نرم‌افزار محافظت می‌کند. هر چند ادعا شده که این نرم‌افزار با استفاده از فناوری پیشرفته از کد در برابر همه دیس‌اسمبلرها و اشکال‌زداها محافظت می‌کند، نسخه قفل‌گشایی شده آن موجود است که خود نشان دهنده قدرت بالای مهندسی معکوس است. ابزار حفاظتی Themida نیز نرم‌افزاری حرفه‌ای و مناسب برای برنامه‌نویسان سیستم عامل ویندوز است

که از دی‌کامپایل کردن کد و قفل‌گشایی برنامه جلوگیری می‌کند.

۲. عدم استفاده از نام‌های رایج برای توابع امنیتی و حفاظتی مانند IsValidSerialNum-ber() و یا Auto-Check()

۳. عدم استفاده از کادر پیام‌ها با متن رایج مانند Thanks for registration و مشابه آن

۴. استفاده از الگوریتم‌های خاص رمزنگاری در برنامه و یا توابعی که فرایند مهندسی معکوس را به تاخیر می‌اندازند.

۵. استفاده از الگوریتم‌های مختلف برای بررسی امنیت نرم‌افزار و اطمینان از دستکاری نشدن آن‌ها

۶. استفاده از روش‌های ضد اشکال‌زدایی به منظور ایجاد اختلال در فرایند تجزیه و تحلیل کد به منظور امنیت بالاتر نرم‌افزار، بهتر است چندین روش از راهکارهای فوق در طی فرایند برنامه‌نویسی رعایت شود.

نتیجه گیری

مهندسی معکوس در حوزه نرم‌افزار، روند بررسی و تحلیل موضوعی از سیستم است که امکان ساخت و نمایش آن سیستم را در سطوح بالاتر فراهم می‌کند. در واقع مهندسی معکوس تجربه دیگران را که یک بار با موفقیت انجام دادند تکرار می‌کند. به صورت کلی این مدل شامل چهار مرحله است. در مرحله اول برنامه هدف تحلیل می‌شود. پس از آن سطح میانی ساخته شده و تجزیه و تحلیل می‌شود. سپس جزئیات کدهای به وجود آمده بررسی شده و در نهایت با توجه به مراحل قبل، محصول جدید ساخته می‌شود. از کاربردهای این روش می‌توان به نوسازی نرم‌افزار، به روز رسانی مستندات، نگهداری محصول و به دست آوردن اطلاعات حساس به کمک تجزیه و تحلیل طراحی اجزای سیستم و کدهای اسمبلی اشاره کرد. ابزار اصلی مورد نیاز برای انجام مهندسی معکوس یک ذهن خلاق می‌باشد که بتواند مفهوم اصلی هر قطعه کد را به سرعت پیدا کند. می‌توان گفت مهمترین بخش فرایند معکوس کردن، فهم کد برنامه است. در این نوشتار ابتدا مهندسی معکوس در حوزه نرم‌افزار تعریف شده و کاربردهای آن گفته شد. سپس مفاهیم بنیادی زبان اسمبلی و ابزارهای مورد نیاز برای معکوس‌سازی معرفی شدند. در نهایت معایب استفاده غیرقانونی آن و قوانین حمایت از حقوق پدیدآورندگان بیان شد. می‌توان نتیجه گرفت مهندسی معکوس در حوزه نرم‌افزار در صورت رعایت موازین قانونی نقش مهمی در پیشرفت صنعت تولید نرم‌افزار دارد و کشف فناوری‌های نهفته

در یک محصول و استفاده از مهندسی معکوس باعث می‌شود پله‌های رشد در کسب دانش نرم‌افزاری را با سرعتی بیشتر طی کنیم.

مراجع

1- E. Eliam, Secrets of Reverse Engineering, Indianapolis, IN: Wiley, 2011.

2- Product Design: Techniques in Reverse Engineering and New Product Development by K.Otto and K. Wood Prentice Hall, 2001.

3- Reverse Engineering: An Industrial Perspective by Raja and Fernandes. Springer-Verlag 2008

4- Reverse Engineering in Computer Applications. MIT Lecture Notes 2006

5- Behrens, Brian C. Reuven Levary, , "Legal Aspects
Software Reverse Engineering and Copyright: Past, Present and Future", John Marshall Law Review, Vol. 31 (1), 1997-98.

6-https://isa.sbu.ac.ir/attachments/article/298/Master7-Intro to Reverse Engineering: http://www.acm.uiuc.edu/sigmil/RevEng/index.html

8-Frequently Asked Ques>ons (and Answers) about Reverse Engineering: http://www.chillingeffects.org/reverse/faq.cgiQID188

9-OllyDbg, Available from: http://www.ollydbg.de/; [Accessed 21-06-2018]

10- B. W Weide, W D. Heym, J. E. Hollingsworth, "Reverse engineering of legacy code exposed," in Proc. 17th Int. Conf. Software Engineering, Seattle, Washington, WA, 1995.

11-A. V. Deursen, J. Favre, R. Koschke, and J. Rilling, "Experiences in

Teaching Software Evolution and Program Comprehension," in Proc. 11th IEEE Int. Workshop on Program Comprehension, Washington, DC, 2003.

واژه‌نامه‌ی فارسی به انگلیسی

الف

اشکال‌زدا ..... Debugger

انباره ..... Stack

انفصال سخت‌افزاری ..... Hardware breakpoint

انفصال نرم‌افزاری ..... software breakpoint

ب

بخشهای قابل اجرای داده ..... Executable Data Sections

پ

پایش سیستم ..... System-monitoring tool

ت

تجزیه و تحلیل بر خط ..... Live code analysis

تجزیه و تحلیل برون خط ..... Offline code analysis

ث

ثبات ..... Register

ج

جریان کنترل ..... Control flow

د

دیس‌اسمبلر ..... Disassembler

دی‌کامپایلر ..... Decompiler

ق

قرارداد بسته داده کاربر ..... User Datagram Protocol

قرارداد کنترل انتقال ..... Transmission Control Protocol

ک

کد عملیاتی ..... Operation code

کد ماشین ..... Object code

م

ماشین مجازی جاوا ..... Java Virtual Machine

معکوس‌سازی در سطح سیستم ..... System-level reversing

معکوس‌سازی در سطح کد ..... Code-level reversing

معکوسسازی مربوط به امنیت ..... Security-related reversing

معکوس‌سازی مربوط به تولید نرم‌افزار ..... Software development related reversing

ه

هرم ..... Heap

Abstract

Traditional software engineering is primarily focused on the development and design of new software. However, most programmers work on software that other people have designed and developed. Reverse engineering is the process of translating compiled programs to source code, and analyzing the resulting code. It is useful, since without knowing the inside of a program it is very difficult to build onto it, create software that can interact well with it, or simply create a similar program of your own. The challenge of reverse engineering lies within the fact that a lot of information contained within the source code of a program is destroyed in the compilation process, and re-obtaining it is done through different kinds of analyses. This paper introduces the terminology of reverse engineering and gives some of the obstacles that make reverse engineering difficult. Although reverse engineering remains heavily dependent on the human component, a number of automated tools are presented that aid the reverse engineer.

College of Science

School of Mathematics, Statistics, and Computer Science

Reverse Engineering of Software

Marzieh Akbarifar

Supervisor: Ebrahim Naghibzadeh Mashayekh

A thesis submitted to Graduate Studies Office

in partial fulfillment of the requirements for the degree of

Master of Science in

Computer science

2018

۵۴