تاریخ دریافت مقاله: ۹۵/۱۲/۰۲
تاریخ پذیرش مقاله: ۹۶/۰۳/۱۲

محافظت از نرم‌افزار با استفاده از مبهم‌سازی

امیر اکبری‌فر*
کارشناس ارشد مهندسی نرم افزار، دانشکده فنی و مهندسی – دانشگاه آزاد دامغان – دامغان – ایران
پست الکترونیکی: msc.akbarifar@gmail.com

رضا مرتضوی
استادیار دانشکده فنی و مهندسی – دانشگاه دامغان – دامغان – ایران
پست الکترونیکی: r_mortazavi@du.ac.ir

چکیده

با پیشرفت فنّاوری دیجیتال، تکثیر غیرقانونی نرم‌افزار، رشد غیرقابل‌تصوری پیدا کرد. ازاین‌رو نرخ سرقت‌های نرم‌افزار نیز به شکل قابل‌توجهی افزایش یافت. فنون متعددی برای محافظت از کد، موجود بوده که یکی از این راهکارها، مبهم‌سازی کد است. مبهم‌سازی کد، سازوکاری برای پنهان‌سازی الگوریتم اصلی، ساختمان داده یا منطق کد، جهت جلوگیری از عملیات مهندسی معکوس غیرمجاز در راستای حفاظت از کد است. اغلب راهکارهای تحلیلی، مادامی‌که توسط انسان مورد کاربرد قرار گیرد، دچار شکست خواهد شد، ازاین‌رو مبهم‌سازی به‌منظور حفاظت از مالکیت معنوی نرم‌افزار و جلوگیری از تشخیص کد مخرب در بدافزارها، همواره چالشی بزرگ به شمار می‌آید. ازآنجاکه هیچ‌کدام از فنون مبهم‌سازی جاری، تمام اثربخشی موردنیاز مبهم‌سازی در مقابله با حملات مهندسی معکوس را اقناع نمی‌سازد، پژوهشگران صنعت نرم‌افزار به دنبال ایجاد بهترین و جدیدترین فنون مبهم‌سازی در جهت نیل به مالکیت معنوی در فرایندی منظم هستند. هدف اصلی این نوشتار، دسته‌بندی فنون تحلیل ایستا و پویای رده‌های متعدد مبهم‌سازی در مقابل فنون مبهم زدایی است. در سویی دیگر، برای مقایسه و یافتن بهترین روش، دسته‌بندی از روش‌های مبهم‌سازی نرم‌افزاری با توانایی حفاظت منطقی از برنامه‌ها در برابر ابزار تحلیل کد بیان‌شده است.

کلید واژه‌ها: مبهم‌سازی نرم‌افزار، مهندسی معکوس، محافظت نرم‌افزاری، بدافزار، استاکس_نت.

۱- مقدمه

صنعت فنّاوری اطلاعات سالانه میلیاردها دلار جهت ممانعت از حملات مخربی همچون دست‌کاری در کد و مهندسی معکوس صرف می‌نماید. با توجه به کاربردهای کلان و نیز توسعه اینترنت ضرورت لازم در پژوهش بر روی امنیت و محافظت از کد ایجادشده است. هر سازمانی مالکیت معنوی داشته و چالش بزرگ پیش روی آن محافظت از داده‌های خود در جهت حفظ حریم خصوصی نرم‌افزار مورداستفاده و یا جلوگیری از نفوذ کد مخرب

* نویسنده مسئول

اســـت. همچنین پردازش داده توســـط درخواســـت، کاملاً محرمانه اســـت؛ بنابراین کشف این فرایند منجر به آسیب مستقیم و جدی به حوزه اقتصاد نرم‌افزار می‌گردد. به‌طور عمومی دو راهکار قانونی و فنی برای حفظ مالکیت معنوی[1] وجـــود دارد. راهکار قانونی بـــه معنای اخذ حق تکثیر و یا تعییـــن قرارداد و امضای قانونـــی در برابر تکثیر مجدد و غیرقانونی است. راهکار فنی بدین معناست که تولیدکننده نرم‌افـــزار، راهـــکاری منحصربه‌فرد را بـــرای محافظت از نرم‌افزار خود ایجاد می‌کند.

در راســـتای مقابله با خراب‌کاری، بهترین ایده استفاده از ســـازوکار امن‌سازی به همراه نرم‌افزار کاربردی است. یکی از انواع این فنون، مبهم‌ســـازی است که حوزه نوینی برای پژوهش و تحقیق در حوزه امنیت نرم‌افزاری در عصر دیجیتال است. مبهم‌سازی متشکل از تبدیل و دگرگون‌سازی کد اســـت که در عین حفظ شدن ویژگی‌های اصلی با تغییر در ســـاختار خود، درک برنامه را برای تحلیلگر دشـــوار می‌نمایـــد. رمزنـــگاری و دیوار آتـــش راهکارهای تدافعی دیگـــری در جهت کاهـــش خطر حملات نفوذگران اســـت. ولیکن، این راهکارها در هنگامی‌که متخاصم به‌عنوان کاربر نهایـــی واقع می‌گـــردد، کمکی برای محافظـــت از نرم‌افزار نخواهند نمود.

در بین روش‌های موجود برای محافظت کد از حملات مختلف، مبهم‌ســـازی کد روشی مشهور در جهت ممانعت از درک کـــد و یا دســـت‌کاری کد به شـــمار می‌آید. روش مذکور تا حد زیادی به همراه راهکارهای متعدد ارائه‌شده، موردقبول واقع گردیده است. همواره موضوع فوق نوعی از محافظـــت نرم‌افـــزاری در مقابله با مهندســـی معکوس غیرمجاز به‌حساب می‌آید. اگرچه ممکن است یک متخاصم مصمّم، بعد از گذراندن مدت‌زمان کافی برای بررسی دقیق کد مبهم‌ســـازی شـــده، فرایند و عملکرد دگرگون‌سازی را معین کـــرده و در هدف مخرب خود موفـــق گردد. به این دلیـــل، فنون مبهم‌ســـازی با راهکارهـــای دیگری همچون به‌روزرسانی و دوباره جایگذاری کد، تشخیص دست‌کاری و بروز رسانی محافظت‌شده در میان سازوکارهای تدافعی اعمال‌شـــده‌اند. روش‌هایی همچون رمزنـــگاری، محافظت در ســـمت ســـرویس‌دهنده، راهکارهای امنیتـــی مبتنی بر ســـخت‌افزار، کدهـــای محلی علامت‌گذاری شـــده مختلف، تصحیح دست‌کاری و نهان‌نگاری پرکاربردترین راهکارها جهـــت اجتناب و ایجاد چالش برای موتورهای تشـــخیص هســـتند. هرچند، فراهم‌کننده می‌بایســـت نسبت به تخمین مقاومت مبهم‌ســـازی (مدت‌زمانی که صرف می‌شـــود تا نفوذگر به درک کد برسد) اقدام نماید. بر این اساس بعضی از روش‌های مبهم‌ســـازی توانایی پیاده‌سازی بر روی کد اصلی را دارا هستند. ازاین‌رو متخاصم توانایی درک منطق و الگوریتم کد را نخواهد داشت.

فنون مبهم‌ســـازی دربرگیرنـــده راهکارهایی همچون دوباره بازســـازی کـــد، دگرگون‌ســـازی بـــرای نام‌های مشـــخص‌کننده معنی‌دار در کد اصلی با عناوین و نام‌های بی‌معنی به‌صورت تصادفی (نام‌گذاری شناســـه)، درج کد مـــرده، پرش‌های فاقد شـــرط و قید، پرش‌های شـــرطی و باقاعده، درج انشـــعاب‌های شفاف، تخصیص متغیر، کد از بین رفته تصادفی، ادغام اعداد تصادفی صحیح، رمزگذاری رشته، تولید جعلی سطح کد میانی، محو ساختن ثابت‌ها، به دام انداختن جریان‌های کنترلی و ... هستند. در سوی دیگر، کد نرم‌افزاری سیار بوده و به‌صورت توزیع‌شده در شبکه وجود دارد که همـــواره به‌عنوان ویژگی غیرقابل‌اعتماد به شـــمارمی‌آید، بنابراین ســـازوکار محافظت بایستی همراه نرم‌افزار و مستقل از سخت‌افزار باشد. بر این اساس، فرض بر این اســـت که مبهم‌ســـازی کد، ابزاری ساده و غالب در بخـــش محافظت از کد منبع در حوزه امنیت و نیز محافظت از نرم‌افزار اســـت. ایـــده و راهکار اصلی فراســـوی فنون مبهم‌ســـازی، پنهان نمودن کد در مقابل متخاصمین است. اگرچه کد دگرگون خواهد شـــد ولی ویژگی‌های آن مشابه کد اصلی خواهد بود، ولیکن برای درک و تحلیل به مشکلی بزرگ بدل خواهد شـــد. ازجمله راهکارهای مبهم‌سازی در جهت محافظت از کد می‌توان به فنونی همچون تبدیل داده،

1- Intellectual Property

بازتاب رقم نقلی، نشانی‌دهی غیرمستقیم، نشانی‌دهی ثبات، ترکیب دستورالعمل‌های دودویی و کد اسمبلی، ترکیب اعداد اعشاری و دودویی با دستورالعمل‌های کد اسمبلی و نیز استفاده از اعداد اعشاری در بین دستورالعمل‌های کد اسمبلی اشاره نمود. [۱].

در اواخر دوره ۱۹۸۰ میلادی، بدافزارها دلیل عمده‌ای برای مقاوم‌سازی فنون مبهم‌سازی کد بودند. برای نمونه در سال ۱۹۸۶، ویروس کامپیوتری برین[۲] که به اولین ویروس کامپیوتری سیستم‌عامل داس[۳] تعلق داشت، عملکرد خود را به‌واسطه محدودسازی کد دودویی ویروس و با تظاهر آن به‌عنوان محتویاتی بی‌ضرر، مبهم‌سازی می‌نمود [۲].

گسترش کاربردهای مبهم‌سازی در صنعت بدافزار، آغازی بر تقابل بین پژوهشگران حوزه نرم‌افزار و تحلیلگران بود. پژوهشگران حوزه امنیت نرم‌افزار، روزبه‌روز فنون پیچیده‌تری را برای پنهان‌سازی رفتار کد، توسعه داده‌اند و این در حالی است که تحلیلگران نیز به‌طور فزاینده‌ای از فنون پیچیده تحلیل کد در جهت غلبه بر مبهم‌سازی استفاده می‌نمایند. در طی دهه گذشته، تشخیص بدافزار تبدیل به مشغله‌ای چند میلیون دلاری و نیز به حوزه‌ای مهم در تحقیقات محیط دانشگاهی بدل شده است. تحلیل ایستا همواره فن غالب برای تشخیص بدافزار (پویشگر ویروس) در سمت کاربر بوده و دچار تغییر زیادی در سال‌های اخیر نگردیده است. پویشگرهای ویروس[۴] فعلی عموماً مبتنی بر سازوکار تشخیص برپایهٔ شناسه[۵] هستند [۳]. از سوی دیگر بدافزارها در سال‌های اخیر به‌طور عمده‌ای تکامل‌یافته‌اند که اغلب از فنون پیچیده سطح بالای مبهم‌سازی برای جلوگیری از تشخیص و دشوارسازی درک کد بهره‌گیری می‌کنند. رمزگذاری، چندریختی[۶] و علاوه بر آن، فراریختی[۷] به‌صورت عمومی به‌منظور شکست و غلبه بر سازوکار تشخیص مبتنی بر امضا توسط پنهان‌سازی ویژگی‌های مخرب در بخش‌های داده‌ای از دودویی که برای هر نمونه بدافزار متفاوت است، گسترش می‌یابد. امروزه اغلب راهکارهای مبهم‌سازی بدافزار از ایده کلی و سادهٔ پنهان‌سازی کد مخرب توسط بسته‌بندی و یا رمزگذاری آن به‌عنوان داده‌ای که توسط ماشین مورد تفسیر قرار نگیرد، استفاده می‌نمایند. در زمان اجرا، روالی برای تبدیل بلوک‌های داده‌ای به کد تفسیری ماشین مورد کاربرد قرار خواهد گرفت. چندریختی و فراریختی برای بهبود بخشی مضمون بسته‌ها با هدف دشوارسازی تشخیص خودکار بدافزار مورد کاربرد قرار می‌گیرند[۴] نوع دیگری از بسته‌بندی توسط وُو[۸] معرفی‌شده است. این راهکار که می‌مورفیسم[۹] نام دارد، کدهای برنامه را به‌عنوان کدهایی در ظاهر بی‌ضرر، رمزگذاری نموده تا با مفاهیم کلی قبلی (تجزیه‌وتحلیل مبتنی بر آنتروپی) قادر به تشخیص نباشد و به‌عنوان کد بسته‌بندی‌شده بر روی کد اعمال می‌شود [۵]. تشخیص و تجزیه و تحلیل بدافزارهای بسته‌بندی شده برای سال‌ها مورد مطالعه و پژوهش قرار گرفته است. بسیاری از راهکارها مبتنی بر تجزیه‌وتحلیل ایستا هستند. کدهای رمزگذاری شده مبتنی بر تحلیل آنتروپی که توسط لیدا و همکاران[۱۰] بیان‌شده، قابل‌شناسایی هستند [۶]. بروسکی[۱۱] راهکاری برای تشخیص بدافزارهای خودتکثیرکننده توسط تطبیق گراف کنترل جریان رویه‌ای نرم‌افزار در مقابله با نمونه‌های بدافزاری تشریح کرد. طراح استدلال کرده است که باوجود ذات خودتکثیرکننده، گراف کنترل جریان این نوع از بدافزارها برای تشخیص آگاهانه به‌اندازه کافی دارای مشخصه‌های سودمند است. به‌طور واضح، برای تشخیص مبهم‌سازی مبتنی بر بسته‌بندی، تحلیل واقعی معنایی کد در نوشتارهای دیگر با راهکارهای مختلف ارائه‌شده است [۷].

مارک استمپ و همکاران[۱۲] در مقاله‌ای به مقایسه تحلیل‌های ایستا، پویا و ترکیبی تشخیص بدافزار با

2- Brain
3- MS-DOS
4- Virus scanner
5- Signature based detection
6- polymorphism
7- Metamorphism

8- Wu
9- Mimimorphism
10- Lyda et al.
11- Bruschi
12- Mark Stamp et al.

اســـتفاده از مدل‌های مخفی مارکـــوف[13] پرداخته‌اند. آن‌ها تحلیل ترکیبی را پیشنهاد نموده‌اند که طی آن تحلیل پویا در مرحلهٔ آزمایش و تحلیل ایســـتا در مرحلهٔ تشخیص به‌کار برده می‌شود. پژوهشگران مذکور، فنون تشخیص بدافزار را مبتنی بر ترتیب فراخوانی‌های واســـط برنامه‌نویســـی کاربـــردی[14] و ترتیب کدهای مربوط به دســـتورالعمل‌های واحد پردازشی[15] تحلیل کرده‌اند .[۸]

رانـــا و اســـتمپ[16]، در پژوهشـــی با عنوان تشـــخیص ســـرقت نرم‌افزاری با تحلیل فراریختی بـــه بحث پیرامون تشخیص سرقت و همچنین اعمال دست‌کاری در نرم‌افزار پرداخته‌اند. آن‌ها فنون متعدد تشخیص را که در مرجع [۸] مطالعه شـــده اســـت تحلیل نموده‌اند. آن‌ها به ازای هر فن، نرخ تشـــخیص را به‌عنوان تابعی از درجه دســـت‌کاری کد اصلی تعیین نموده‌اند. نتایج آن‌ها بهبود قابل‌توجهی را در مقایســـه با سایر پژوهش‌های مشـــابه در پی داشته است. راهکار آن‌ها قابل‌اعمال در نرم‌افزارهای در دســـت اســـت و نیاز به دسترســـی کد اصلی ندارد. فـــرض آن‌ها بر این بود که نرم‌افزار سرقت شـــده، جهت اجتناب از تشخیص، برنامه‌ریزی‌شده است. آن‌ها فرایند اصلاح را با بهره‌گیری ایده‌ای از مبحث تشخیص بدافزار چندریختی شبیه‌سازی نموده‌اند[۹] .

دا لیـــن[17] و همچنین مارک اســـتمپ، در پژوهشـــی به تحلیل بدافزارهای فراریختی غیرقابل تشخیص پرداخته‌اند. بر طبـــق نظر آن‌ها ســـازوکار تشـــخیص پویشـــگرهای ضدویروس تجاری عموماً مبتنی بر شناسه است. ازاین‌رو، تنها برای الگوهای شناخته‌شـــده جهت تعیین آلودگی فایل مورد کاربرد قـــرار می‌گیرد. برای ممانعت از تشـــخیص مبتنی بر شناسه، طراحان بدافزار از فنون مبهم‌سازی جهت ایجاد بدافزارهای چندریختی استفاده می‌کنند. بدافزارهای چندریختی ساختار داخلی خود را مرتباً تغییر می‌دهند، این فرایند گام تدافعی مهمی در راســـتای تشـــخیص مبتنی بر شناسه است [۱۰] .

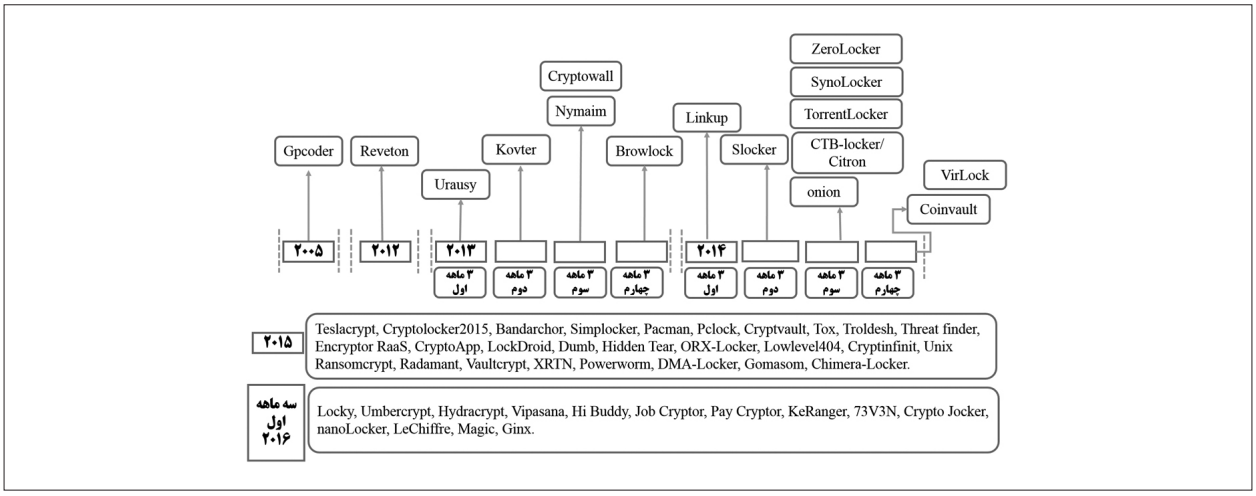
وابستگی اقتصاد زیرزمینی به بدافزار و رشد بی‌رویه آن در وب تیـــره[18]، یکی از ضرورت‌های حوزه توســـعه و پژوهش پیرامون راهکارهای مبهم‌سازی، پدافند غیرعامل و محافظـــت از نرم‌افزارها در مقابله با مهندســـی معکوس غیرمجاز اســـت. انگیزه‌های مهندسی معکوس بر روی کد متفاوت است. در یک ســـناریوی مخرب، تحلیلگر با توجه به علاقه خود نســـبت به استخراج اطلاعات پنهانی همچون کلیـــد پنهان رمزنـــگاری و الگوریتم برنامـــه از کد برنامه اقدام می‌نماید. ازاین‌رو پژوهشـــگران حوزه امنیت، شاهد گسترش بدافزارهای چندریختی و فراریختی در چند سال اخیر هستند. از مشهورترین کاربردهای بدافزارهای اخیر می‌توان به باج‌افزار قفل نمودن سیســـتم[19]، حمله باج‌افزار بـــه رکورد راه‌انداز اصلی[20]، باج‌افزار رمزگذاری کارســـاز وب[21] و باج افزارهای تلفن‌های هوشـــمند[22] اشـــاره نمود. اگرچه باج افزارها به شـــکل عمومی در سال ۲۰۰۵ توسط جی‌پی‌کدر[23] پا به عرصه ظهور نهاده‌اند ولی گستره کاربرد آن‌ها در بین ســـال‌های اخیر بوده اســـت. شکل ۱ – بیانگر رشد بدافزارهای گوناگون در بین سال‌های ۲۰۰۵ تا ۲۰۱۷ میلادی است.

هدف اصلی این نوشتار، فراهم نمودن دیدگاهی جامع پیرامون فنون مبهم‌ســـازی کد و نیز ســـنجش کارایی آن در مقابل مبهم زدایی و عملیات مهندســـی معکوس اســـت. از طرفی کمبود منابع شفاف در حوزه مبهم‌سازی همواره چالشـــی بزرگ محسوب می‌شود. ازاین‌رو در این نوشتار به‌طور انتزاعی به دسته‌بندی فنون مبهم‌سازی و همچنین تعریف جامع روش‌های مذکور پرداخته‌شـــده است. برای نیل به این هدف، طبقه‌بندی از ســـناریوهای تحلیلی از دید مبهم کننده و مبهم زدا گردآوری‌شده است [۱۱،۱۲،۱۳].

در این نوشـــتار، به بررســـی توانایـــی حفاظت منطقی

13- Markov hidden models
14- Application programming interface (API)
15- Upcode
16- Hardikkumar Rana and Mark Stamp
17- Da Lin

18- Dark web
19- Locker ransomware
20- Master boot record ransomware
21- Web server encryption ransomware
22- Mobile device ransomware
23- Gpcoder



شکل ۱: رشد بدافزارهای خودتکثیر کننده در طی سال‌های ۲۰۰۵ تا ۲۰۱۶ میلادی

روش‌هـــای بـــه‌کار رفتـــه در برنامه‌ها در برابـــر فنون و ابزارهای تحلیل کد پرداخته‌شده است. علیرغم بیش از دو دهه مطالعه موردی بر روی نظریه‌های مبهم‌سازی، تاکنون مفاهیم قابل‌اطمینانی برای ارزیابی فنون مبهم‌سازی ارائه نشده اســـت. از سویی دیگر، محدودیت‌های مشابهی برای ارزیابـــی، تحلیل کـــد و همچنین فنون مبهـــم زدایی وجود دارد. با در نظر گرفتن این محدودیت‌ها، بازخوانی از فنون تحلیل کد در برابر رده‌های متعدد مبهم‌ســـازی در حملات بخصوص گردآوری‌شـــده است. این نوشـــتار با استفاده از پژوهش‌هـــای فوق‌الذکر، اقدام به معرفی و دســـته‌بندی فنون مبهم‌ســـازی با در نظر گرفتن مزایا و معایب هر فن مبهم‌سازی نموده است. رویکرد این نوشتار برای طراحان نرم‌افزاری و همچنین طراحان بدافزار کاربرد فراوانی دارد. نویســـنده بدافزار با استفاده از روش مبهم‌سازی بهینه در جدول ۳ قادر به پنهان‌ســـازی کد مخرب و از سویی دیگر، طراحان نرم‌افزاری قادر به پنهان‌ســـازی کدهای حساس واصلی برنامه‌ها خواهند بود.

در ادامه این نوشـــتار بخش دوم به مهندسی معکوس، بخش سوم به ســـناریوهای تحلیلی ایســـتا و پویا، بخش چهـــارم بـــه مبهم‌ســـازی نرم‌افـــزاری، بخـــش پنجـــم به تحلیـــل کد و در بخش ششـــم بـــه نتیجه‌گیـــری پرداخته شده است.

۲- مهندسی معکوس

کامپیوترها تنها یک‌زبان را تشخیص می‌دهند و آن‌هم زبان ماشین است، درنتیجه تمامی برنامه‌های نوشته‌شده به زبان ســـطح بالا، باید برای اجرا به زبان ماشـــین تبدیل شـــوند. ازاین‌رو دســـتورات برنامه ابتدا در یک فایل متنی به نام کد منبع درج‌شـــده و ســـپس توسط مترجم به زبان ماشـــین ترجمه خواهد شـــد. حاصل این ترجمه یک فایل اجرایی[۲۴] بوده که روی بُن‌سازهٔ[۲۵]مناسبی اجرا خواهد شد. در زبان‌هایی مثل سی یا وی‌بی[۲۶]، دستورالعمل‌های برنامه مســـتقیماً به زبان ماشین تبدیل می‌شوند، اما در زبان‌های مبتنی بر دات نت و جاوا، دســـتورات هنگام ترجمه شـــدن ابتـــدا به کد میانی که واســـط بین زبان برنامه‌نویســـی و بُن‌سازه است، تبدیل گشته و سپس کد ماشین از روی آن تولید می‌شـــود. حال یک تحلیلگر بدون داشتن کد منبع، به کدهای برنامه دسترســـی پیداکرده و آن‌ها را تغییر خواهد داد، یعنی با اســـتفاده از یک فایل اجرایی یا کد میانی به کد منبع خواهد رســـید. به این عمل مهندســـی معکوس[۲۷] گفته می‌شود. سطوح مهندسی معکوس را می‌توان به دسته‌های مهندسی معکوس در سطوح کد دودویی، متن برنامه و داده تقسیم نمود.

مهندســـی معکوس راهکاری برای استخراج مفاهیم از

24- Executive file (exe)
25-Platform
26- C or VB language
27- Reverse engineering

متن برنامه‌ها است که فراهم‌کننده موارد زیر است.

تبدیـــل کد دودویی توســـط برگردان‌ها[28] بـــه کد زبان ماشین

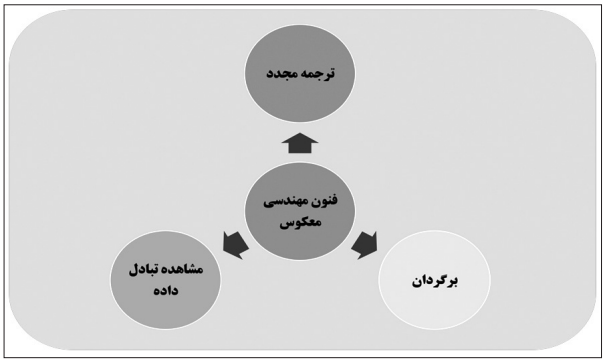
- تبدیل کد دودویی توسط بازترجمه[29] به کد متن
- استخراج مدل‌های رفتاری و ساختاری از متن برنامه
- استخراج منطق برنامه توسط ابزاری همچون دایکون
- بهینه‌سازی مجدد کد برنامه‌ها
- ارزیابی کیفی کد برنامه‌ها
- بررسی آسیب‌پذیری برنامه‌ها
- تشخیص سرویس‌های برنامه و تبدیل به وب‌سرویس

۲-۱. فنون مهندسی معکوس

فنون مهندسی معکوس به‌واسطه انسان از دسته حملات مضر در جهت حفظ امنیت کد و یا بالعکس در آشکارسازی بدافزارها به شمار می‌آیند. ازاین‌رو دسته‌بندی از این فن‌ها که در شکل ۲ بیان‌شده، به‌طور خلاصه مطرح گردیده است.

- مشاهده تبادل داده: در این روش با استفاده از تحلیلگر گذرگاه و یا ضبط‌کننده بســـته‌ای، داده‌ای که بین نرم‌افزار و ســـخت‌افزار در حال تبادل اســـت توسط تحلیلگر تحلیل می‌شود.
- ترجمه مجدد: در این فن از نرم‌افزارهای خاصی به‌نام برگردان استفاده می‌شـــود. این نرم‌افزار یک فایل اجرایی (کد دودویی) یا کد میانی را به زبان سطح بالا تبدیل می‌کند. یعنـــی از طریق ایـــن نرم‌افزار می‌توان به کـــد منبع برنامه دسترسی پیدا کرد.
- برگردان: در این روش با استفاده از برنامه برگردان، کد ماشین به کد اســـمبلی تبدیل می‌شود. چنانچه تحلیلگر به زبان ماشـــین تسلط داشته باشـــد می‌تواند از روی کد ماشین به طراحی و نحوه انجام محاسبات برنامه پی ببرد. همان‌طور که ذکر شد در برگردان‌ها ورودی کد دودویی و خروجی کد ماشین است. برگردان‌ها همانند واحد پردازش مرکزی، کد دودویی نقطه شـــروع را دریافت و نســـبت به تشـــخیص دستورالعمل‌ها اقدام نموده و آن‌ها را به مرحله

28- Disassembler
29- Decompiler

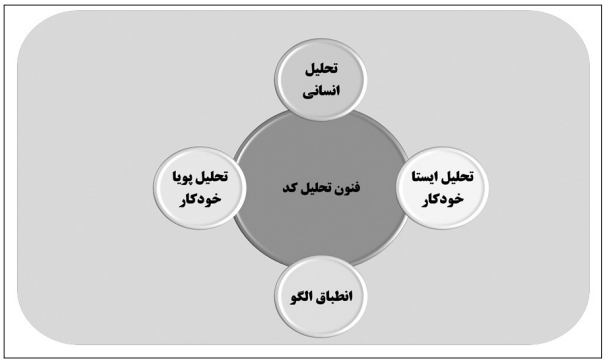


شکل ۲: فنون مهندسی معکوس

اجرا درمی‌آورند. فنون ضد برگردان، عامل تدافعی موفقی در حفاظت از کد، علیرغم وجود گراف کنترل جریان است [۱۱,۱۲,۱۳]. تولیدکننـــدگان محصـــولات نرم‌افزاری، برای ایمن‌سازی کد برنامه و مقابله با حملات مهندسی معکوس و استخراج الگوریتم‌های نرم‌افزاری، فنون مختلفی را به‌کار می‌گیرند. آسان‌ترین و ارزان‌ترین راهکار، مبهم‌سازی کد اســـت. در این روش، منبع برنامه یا کـــد میانی به‌گونه‌ای تغییر داده می‌شود تا ترجمه مجدد آن (بدون تغییر عملکرد در برنامه) بسیار دشوار گردد.

۳- سناریوهای تحلیلی

در این بخش نسبت به تشریح فنون به‌کار رفته در این نوشتار اقدام شـــده است. در این مقاله، سناریوهای ممکن تحلیل بیدرنگ برنامه‌ها در حضور مبهم‌سازی کد، مبتنی بر تحلیل دقیق و مهندسی معکوس ایجاد گردیده است. برای این دسته‌بندی از سناریوهای تحلیلی، متشکل از فن تحلیل و هدف مورد انتظار تحلیلگر برای دسته‌بندی استفاده شده است. فرض نوشتار بر این است که توسعه‌دهنده نرم‌افزاری بیشینه حد ممکن مبهم‌سازی را در ردهٔ بخصوصی تعریف می‌نماید. در ســـویی دیگر، فرض کاملاً مشابهی در سمت تحلیلگر ایجادشده که بیشینه استحکام مبهم‌سازی ممکن در مواجهه با چهار طبقه‌بندی تحلیل کد تعریفی در بخش ۳-۱، ارائه گردیده است. راهکارهای ایجاب‌کننده حوزه کاربردی مبتنی بر فنون متعدد، تحت عنوان اولین طبقه از تحلیل قرار می‌گیرند. برای نمونه، انطباق الگو در کدهای برگردان‌شده،



شکل ۳: فنون تحلیل کد

مستلزم حداقل عملیات موفق برگردانی تحت تحلیل ایستا است. فنون ترکیبی از جنبه‌های متعددی در طبقه‌بندی‌های اشتراکی خاصی نگه‌داری می‌شوند. مقاوم‌ترین طبقه‌بندی، تحلیل به‌واسطه عامل انسانی و نرم‌افزارهای تحلیلگر است که از فنون و نیز انواع مختلفی استفاده می‌کند. نتایج ارائه‌شده تماماً مبتنی بر بیان و تفسیر بوده و از نتایج تحلیلی، استخراج‌نشده است.

۳-۱. طبقه‌بندی تحلیل کد

همان‌طور که در شکل ۳ بیان‌شده، فنون تحلیل کد در چهار ردهٔ عمومی طبقه‌بندی می‌شوند. ازاین‌رو تحلیلگر توانایی استفاده از تحلیل‌های مختلفی را وابسته به هدف خود، نتایج در دسترس و نیز زمان داراست. شکل ۳ بیانگر طبقه‌بندی تحلیل کد است.

• انطباق الگو[30]: فن تطبیق الگو ساده‌ترین و نیز سریع‌ترین حالت از تحلیل کد است که در سطح دودویی ایجاد می‌گردد. فنون این رده دربرگیرنده شناسایی روال ایستای دستورالعمل‌هایی همچون عبارات شرطی و یا طبقه‌بندی‌های مبتنی بر یادگیری ماشین برای داده دودویی است.

• تحلیل ایستای خودکار[31]: در تحلیل ایستا برنامه در سطح دودویی، بدون اجرای آن برنامه بازبینی می‌شود. در تضاد با تطبیق الگوی نحوی، تحلیل ایستا دلایلی پیرامون برنامه معنایی را داراست. در حالت ساده، از برگردان برای تحلیل ایستا استفاده می‌شود که برای تفسیر هدف‌های واحدی کاربرد دارد. از تحلیل ایستا می‌توان به‌دفعات برای بازسازی نمایش اطلاعات سطح بالای برنامه همچون گراف جریان کنترلی استفاده کرد.

• تحلیل پویای خودکار[32]: در تحلیل پویا، برنامه برای مشاهده فعالیت‌ها و جمع‌آوری جریان اطلاعاتی اجرا می‌شود. یکی از مزیت‌های تحلیل پویا، آگاهی دقیق از روند رفتاری برنامه به‌واسطه پایش ردیابی‌های رفتاری است. بااین‌حال، داده جمع‌آوری‌شده از یک یا چندین مرتبه اجرای برنامه، اجازه نتیجه‌گیری پیرامون رفتار داخلی برنامه را نمی‌دهد.

• تحلیل انسانی: در تحلیل به‌واسطه انسان، روند رفتاری برنامه با استفاده از ابزار بخصوصی تحلیل می‌شود. به‌طور عمومی این فرایند تحت عنوان مهندسی معکوس یادشده که در آن تحلیلگر به‌واسطه استفاده از ابزاری خاص و باهدف درک ساختار برنامه و نیز رفتار آن، تلاش در نفوذ به کد برنامه را با مقصودی خاص دارد [۱۱,۲].

۳-۲. اهداف تحلیلگر

مطابق شکل ۴ در تحلیل کد اهداف تحلیلگر به‌منظور ارتقاء پیچیدگی، ازنظر سامانمندی و مشخصه‌های انگیزشی به چهار دسته تقسیم می‌شود.

یافتن محل داده: تحلیلگر همواره تلاش در بازیابی بعضی از داده‌های جای‌دهی شده در نسخه اصلی برنامه را دارد. برای تسهیل در درک می‌توان به استخراج کلید رمزنگاری پنهان، جهت رمزگشایی داده و یا در سناریوهای دیگر، حمله به سازوکار مالکیت معنوی نرم‌افزار و گسترش مجوزهای برنامه، شناسه‌ها و یا داده‌های پیکربندی دستگاه اشاره نمود [۱۱].

تحلیل و یافتن محل اجرایی و عملکرد برنامه: در این کاوش، تحلیلگر تلاش در تشخیص نقطه درونی تابع بخصوصی از برنامه مبهم‌سازی شده را دارد. در سناریوهای اجرایی، تحلیلگر متعاقباً قصد در یافتن نقطه ورودی الگوریتم رمزنگاری در برنامه مبهم‌سازی شده را

30- Pattern matching
31- Automated static analysis
32- Automated dynamic analysis

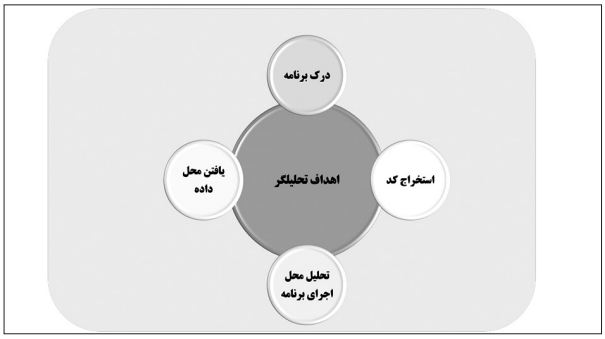
داشــته و یا در ســناریویی دیگر، تحلیلگر به کاوش محل دقیق رونوشتی از سازوکار محافظتی برای سوءاستفاده و یا دست‌کاری رفتاری برنامه را خواهد داشت. به هر جهت، بازنمایی عملکرد بخش بخصوصی از برنامه برای عملیات مهندسی معکوس به‌واسطه انسان بسیار ارزشمند است.

اســتخراج قطعات کد: در این بخش، تحلیلگر تلاش در استخراج قطعه‌ای از کد که دربرگیرنده تمامی وابستگی‌های ممکن و عملکرد بخصوصی از برنامه مبهم‌ســازی شــده اســت را دارد. در سناریو اجرایی، هدف تحلیلگر استخراج الگوریتم رمزنگاری به‌منظور بازســازی روال پیش‌فرض رمزگشایی است. برای نیل به این هدف، نیاز به درک کامل کد نیست.

درک برنامه: عملیــات درک برنامه به تحلیلگر توانایی فهم قطعه‌ای از کد و یا حتی برنامه مبهم شــده را می‌دهد. این فرایند مستلزم این است که تحلیلگر قادر به ابهام‌زدایی و نیز درک کلی برنامه اصلی باشــد. در ســناریو اجرایی، تحلیلگر تلاش بــر درک چگونگی جای‌دهی کلید در برنامه مبهم‌سازی شــده و در راســتای شــروع عملیات تحلیل رمز را دارا است. از ســناریوهای دیگر، می‌توان به یافتن آسیب‌پذیری در جهت تصحیح جریان در نرم‌افزارهایی با عدم دسترس‌پذیری کد و نیز تولید نرم‌افزاری جدید اشاره نمــود. ازاین‌رو اغلب حملات مهندســی معکوس تلاشــی مخاطره‌آمیز در راســتای مالکیت معنــوی نرم‌افزار و نیز تلاشی ثمربخش در بحث تحلیل بدافزار است.

۳-۳. سناریوهای تحلیل ایستا و پویا

ترکیب فنون تحلیلی و اهداف تحلیلگر، سناریوهایی جدید را پدید می‌آورد. در این راستا، تطبیق الگو به تشخیص محل قرارگیری قطعه کد کمک نموده و اســتخراج آن، مســتلزم تحلیل ایستا برای تحلیل وابستگی‌ها بوده که به تقسیم‌بندی دیگری بدل خواهد گشت. شــایان‌ذکر است که تطبیق الگو به‌تنهایــی، درک کد را ممکن نخواهد ســاخت؛ بنابراین این فن مستلزم راهکارهای ترکیبی با سناریوهای تحلیلی است. شکل ۵ بیانگر سناریوهای تحلیل ایستا و پویا است.



شکل ۴: اهداف تحلیلگر

• تشــخیص محــل داده بــا اســتفاده از تطبیــق الگو: الگوریتم تطبیق دهی، جهت تشخیص مکان داده‌ای مطابق خصوصیات الگو به‌کار برده می‌شود. الگوها برای تشخیص خودکار داده درون برنامه، ساختار داده‌ای همچون طول و یا نوع داده را بازنمایی می‌کنند.

• تشخیص محل کد با استفاده از تطبیق الگو: بخش‌های بخصوصــی از کد به‌واســطه تطبیق الگو تشــخیص داده می‌شوند. مادامی‌که تحلیل الگو به خروجی تحلیل ایستا و پویا اعمال شود، تحلیل‌ها مکرراً به ایستا و پویا طبقه‌بندی می‌شوند.

• تشــخیص داده بــا اســتفاده از تحلیــل ایســتا: این تحلیل، برنامه معنایی را برای مشــخص کردن مکان داده بخصوصی تفسیر می‌نماید.

• تشخیص کد با اســتفاده از تحلیل ایستا: تحلیل ایستا توانایــی فراخوانی کد در زمان اجرا را تعیین می‌کند. برای مثال، برگردان‌ها توانایی تفســیر دستورالعمل‌های جریان کنترلی رادارند.

• اســتخراج کد با اســتفاده از تحلیل ایســتا: فن تحلیل ایســتا، وابســتگی‌های قطعه‌ای از کد را که توانایی اجرا و استخراج توسط خود را داشته، تحلیل می‌نماید. اغلب فنون تحلیل ایستا، نتیجه‌ای ثمربخش در استخراج مجموعه‌ای از کدهای موردنیاز را در پی‌دارند.

• درک کد به‌واســطه تحلیل ایســتا: این ســناریو فنون ابهام‌زدایی ایســتا را که توانایی تبدیل قطعه‌ها و همچنین کل کد مبهم‌سازی شده را به بازنمایی کد در سطوح بالا و پایین داشته، ضبط می‌کند. این فرآیند توانایی فهم عملکرد

شکل ۵: سناریوهای تحلیلی ایستا و پویا

کد را به عامل انسانی در طی سناریو حمله مهندسی معکوس می‌دهد.

• کاوش محل داده با استفاده از تحلیل پویا: تحلیل پویا، در دسترس‌پذیری داده را رؤیت نموده و یا توسط برنامه در زمان اجرا، داده را نگه‌داری می‌نماید و از این مشاهدات جهت تعیین داده‌ای بخصوص برای مثال فراخوانی سیستمی و یا محل بخصوصی از حافظه بهره می‌برد.

• تعیین محل کد به‌واسطه تحلیل پویا: تحلیل پویا به رفتار برنامه و تعامل آن با محیط برای مثال فراخوانی‌های سیستمی در زمان اجرا می‌پردازد. در این سناریو، عملکرد بخصوصی از برنامه به‌واسطه رفتاری خاص در زمان اجرا، مشخص می‌شود.

• استخراج کد به‌واسطه تحلیل پویا: تحلیل کد، محل قرارگیری بخشی از کد را تعیین نموده و همچنین وابستگی آن را با ردیابی‌های انجام‌شده، تعیین می‌نماید. استخراج کد با یک‌مرتبه اجرا، ممکن است تمامی اطلاعات لازم برای اجرای دوباره برنامه را در پی نداشته باشد. نتایج گراف کنترلی، زیرمجموعه‌ای از تمامی مسیرهای اجرایی ممکن است.

• درک کد به‌واسطه تحلیل پویا: این سناریو به‌طور خودکار، به فنون مبهم‌زدایی با توانایی تبدیل کد مبهم‌سازی شده به بازنمایی که طی آن تحلیلگر توانایی درک عملکرد برنامه را خواهد داشت، می‌پردازد.

• تحلیل به‌واسطه انسان: راهکارهای ایستا و پویا که به کمک انسان ایجاد می‌شود، هدف در درک کلی جنبه‌های بخصوصی از برنامه را داراست.

۴- مبهم‌سازی نرم‌افزاری

در این بخش به‌طور خلاصه به تشریح راهکارهای مبهم‌سازی و طبقه‌بندی آن‌ها به سه دسته مبهم‌سازی داده (ضبط داده، رمزگذاری، تبدیل داده ایستا به روال‌ها)، بازنویسی ایستای کد (جایگزینی دستورالعمل‌ها، گزاره مات، تزریق کد بی‌استفاده، تزریق کد نامربوط، مرتب‌سازی مجدد، تبدیل حلقه، بازترکیب و قطعه‌قطعه سازی توابع، نام مستعار، درهم‌سازی[۳۳] نام، مبهم‌سازی جریان کنترلی، موازی‌سازی کد، محذوف ساختن فراخوانی‌های کتابخانه‌ای، شکست و محذوف ساختن ارتباط‌ها) و بازنویسی پویای کد (بسته‌بندی یا رمزگذاری، دست‌کاری پویای کد، نیازمندی‌های محیطی، مبهم‌سازی کد به‌واسطه سخت‌افزار، مجازی‌سازی، فنون ضد اشکال‌زدا و برگردان کننده) پرداخته‌شده است. این راهکارها اغلب در آغاز پیدایش در نمونه‌های بدافزاری اعمال گردیده‌اند. مطابق جدول ۲ در ادامه نسبت به تشریح این سه بخش و راهکارهای آن‌ها اقدام شده است.

٤-١. مبهم‌سازی داده

فنون مبهم‌سازی کد به‌منظور سخت سازی فرایند تحلیل کد، مورد کاربرد قرار می‌گیرند. فنون متعددی پیرامون مبهم‌سازی وجود دارد[۱]. ازاین‌رو، چهار فن کلی مبهم‌سازی داده وجود دارد.

مرتب‌سازی مجدد داده: متغیرها همواره توانایی تقسیم به دو و یا چندین بخش را برای دشوارتر نمودن فرایند تحلیل دارند. نگاشت بین مقدار واقعی متغیر و بازنمایی قطعه‌بندی شده آن به‌واسطه دو تابع، یک‌بار در اجرا و در زمان مبهم‌سازی و دیگری در بازسازی مقدار اصلی یک متغیر از اجزای قطعه‌بندی شده در زمان اجرا مدیریت می‌شود. برای نمونه، متغیرهای بولی در میان اجزای متغیر بازیابی می‌شوند. انواع دیگر داده‌ای همچون اعداد صحیح و متغیرهای رشته‌ای در سناریو مشابهی مبهم‌سازی می‌شوند.

33- Scrambling

ساختار آرایه، توانایی تقسیم به زیر آرایه‌ها را داراست. در حقیقت آرایه‌های متعدد توانایی ادغام در یک آرایه رادارند. فنونی همچون درهم‌آمیزی و ایجاد پوشش ازجمله راهکارهای مشابهی است که توانایی نگه‌داری داده مبهم‌سازی شده در آرایه را دارد.

مبهم‌سازی ساختار داده؛ به‌واسطه مرتب‌سازی مجدد اجزا در جهت کاهش موقعیت مکانی، فنی دیگر از مجموعه فنون کلی مبهم‌سازی است. با هدایت دسترسی حافظه، ترتیب داده در حافظه به‌صورت دوره‌ای تغییر می‌یابد و تحلیل آن را بسیار دشوار می‌سازد.

رمزگذاری: داده‌های ایستا (همچون رشته‌ها) به همراه کدهای دودویی، محتوی اطلاعات پرکاربردی در سمت تحلیلگر است. در جهت کاهش نیاز به نگه‌داری داده به‌طور ایستا، داده به بازنمایی‌های مختلفی به همراه تابع بخصوص رمزگذاری در متن ساده[34] به حالت دودویی تبدیل می‌شود. از سویی دیگر در زمان اجرا، تابع معکوسی برای رمزگشایی داده مورد کاربرد واقع می‌گردد.

تبدیل داده ایستا به روال: این فن، داده ایستا را با تابعی که داده را در زمان اجرا محاسبه کرده، جایگذاری می‌کند. برای مثال، شیء رشته‌ای در زمان اجرا ساخته می‌شود؛ بنابراین، تحلیلگر قادر به استخراج مقادیر با استفاده از محاسبه اعداد دودویی نخواهد بود. نوع دیگری از این راهکار مبهم‌سازی، رمزنگاری جعبه سفید است. ایده پایه، ادغام کلید مخفی با عناصر رمزی است. ازاین‌رو، کلید در کد دودویی غیرقابل‌دسترس خواهد بود.

٤-٢. بازنویسی کد به‌صورت ایستا

دوباره‌نویسی ایستا مشابه عملیات بهینه‌سازی در مترجم بوده که کد برنامه را در طی مبهم‌سازی دست‌کاری می‌نماید. در این میان به خروجی اجازه می‌دهد که بدون دست‌کاری‌های بیشتر اجرا شود. در مبهم‌سازی بدافزاری واژه فراریختی، جهش خودکار کدهای دودویی از فنون بازنویسی ایستا را به بازنمایی برگردان‌شده تشریح می‌نماید. دراین‌بین، فنون متعددی همچون جایگزینی دستورالعمل‌ها، گزاره مات، تزریق کد بی‌استفاده، تزریق کد نامربوط، مرتب‌سازی مجدد، تبدیل حلقه، بازترکیب و قطعه‌قطعه سازی تابع، نام مستعار، درهم‌سازی نام، مبهم‌سازی جریان کنترلی، موازی‌سازی کد، محذوف ساختن فراخوانی‌های کتابخانه‌ای، شکست و محذوف ساختن ارتباط‌ها در این دسته‌بندی قرارگرفته شده که به‌اختصار به آن پرداخته‌شده است.

جایگزینی دستورالعمل‌ها: هرگونه رفتار برنامه، توانایی پیاده‌سازی درراه‌های متعددی را داراست و دستورالعمل‌ها یا توالی از دستورالعمل‌ها توانایی جایگذاری‌های مختلف نحوی را دارا هستند. برای مثال در بُن‌سازه اینتل x86، و در دستورالعمل MOV EAX, 0 و XOR EAX، مقدار EAX معادل بوده و توانایی جایگذاری با یکدیگر را دارد. فنون مبهم‌سازی پیچیده‌تر از این نوع دربرگیرنده جایگذاری فراخوانی‌ها با ترکیبی از دستورالعمل‌های PUSH و RET را داراست. جایگذاری به‌طور غیر مکرر از «کدهای» با بلوک‌های مورد کاربرد، جهت کاهش تعداد کل کدهایی مختلف مورداستفاده و نیز برای نرمال‌سازی دامنه کاربرد آن‌ها، به‌کار می‌رود. به‌طور مشابه، کد پوسته[35] در جهت استخراج توانایی تبدیل به روالی بی‌ضرر از نویسه‌های ورودی توسط ابزارهایی همچون اس سی ام مورفیسم[36] را داراست. کدهای مخرب نیز توانایی پنهان گشتن در روالی بی‌ضرر از نویسه‌ها رادارند

گزاره مات: یک گزاره (تابع بولی)، در صورتی مات اطلاق می‌شود که خروجی آن برای مبهم‌ساز در زمان مبهم‌سازی شناخته‌شده باشد. گزاره مات برای پیچیده نمودن فرایند مهندسی معکوس به‌کار می‌رود. برای نمونه، مبهم‌سازی گراف کنترل جریان برنامه به‌واسطه اضافه نمودن پرش‌های شرطی که به نتیجه گزاره‌های مات وابسته است، ایجاد می‌گردد.

تزریق کد مرده: واژه کد مرده یا کد بدون جان، به

34- Plaintext

35- Shell code
36- SCMMORPHISM

جدول ۱: درج کد مرده [۹]

اضافه نمودن کد هرز	کد اصلی
ADD 1058, EAX JMP loc1234 POP EBX PUSH EBX PUSH EBX POP EBX loc1234 SUB EAX	ADD 1055h, EAX SUB EAX

بلوک‌هایی اشـــاره نموده که به‌طور ساده در گراف کنترل جریان در دســـترس نیست و هرگز اجرا نخواهند شد. این کدها تحلیل برنامه را از حیث زمانی با افزایش کد دشوارتر می‌نمایند. درج کد مرده یکی از فنون رایج به‌کار گرفته‌شده در موتورهای چندریختی است. در این روش، مجموعه‌ای از بایت‌هـــا به‌واســـطه درج کد مرده تغییر داده می‌شـــود. دستورالعمل‌های به‌کار رفته در کد مرده، هیچ‌گونه تأثیری بر کد اصلی ندارنـــد. کدهای مرده درج‌شـــده، هرگز اجرا نخواهند شـــد و ازاین‌رو هیچ‌گونه تأثیـــر معنایی درروند اجرایی برنامه نخواهد گذاشت. این راهبرد، برای جلوگیری از فرایندهای تشخیص مبتنی بر شناسه و مبتنی بر تحلیل آماری، مورداستفاده قرار می‌گیرد. جدول ۱ بیانگر درج کد مرده است.

تزریق کد هرز: روال دســـتورالعمل‌هایی که تأثیری بر اجرای برنامه نداشـــته، توانایی درج در کد را برای پیچیده نمودن فرایند تحلیل کد، داراســـت. ساده‌ترین حالت ممکن در فن فوق‌الذکر، دستورالعمل‌های NOP بوده که وضعیت برنامه را دست‌کاری نمی‌کند. در مقایسه با درج کد مرده، کدهای نامربوط به‌واسطه اجرای جریان کنترلی برنامه در زمان اجرا، در دسترس خواهند بود.

مرتب‌ســـازی مجـــدد: مشـــابه ســـاختار داده همواره شـــرط‌ها و جملات، توانایی باز مرتب‌ســـازی را در جهت کاهش موقعیت خود داشـــته و این ترتیب اختلالی درروند اجرایی برنامه تولید نخواهد کـــرد. فنون دیگر، در حقیقت برای بهینه‌ســـازی کد معرفی گردیـــده ولیکن تنها در متن مبهم‌ســـازی از آن استفاده می‌شـــود. این مفاهیم حتی در صورت تقســـیم‌پذیری به اجزای دیگر، عملکردی یکســـان همانند بدافزار «استاکس‌نت»[۳۷] دارند.

تبدیـــل حلقـــه: تبدیل‌های متعـــددی در جهـــت افزایش پیچیدگی کد وجود داشـــته که آن‌ها را برای اســـتفاده در مبهم‌ســـازی برجســـته می‌نماید. حلقه‌ها در حقیقت برای بهینه‌سازی رفتار کد، مورد کاربرد قرار می‌گیرند.

بازترکیب و قطعه‌قطعه سازی: شبیه‌سازی تابع بیانگر مفهـــوم قطعه‌قطعه ســـازی جریان کنترلـــی در دو یا چند مسیر مختلف است که در عین هم ارزی از منظر تحلیلگر، متفاوت خواهد بود.

٤-۳. بازنویسی کد به‌صورت پویا

مشـــخصه اصلی راهکارهای مبهم‌ســـازی کد در این طبقه‌بندی این است که کد اجرایی متفاوت از کدی است که به‌طور ایستا در اجرا قابل‌مشاهده است. روش‌هایی همچون رمزنگاری، دســـت‌کاری پویای کد، نیازمندی‌های محیطی، مبهم‌سازی کد به‌واسطه سخت‌افزار، مجازی‌سازی و فنون اشکال‌زدایی[۳۸] و برگردان از زمره این راهکارها است.

رمزگذاری: در مقوله بدافزار، راهکارهای مبهم‌سازی متعددی درزمینه بســـته‌بندی و رمزنگاری بسته وجود دارد. ایـــن راهکارها که موجب پنهان‌ســـازی کد بدافزار می‌شوند به‌واسطه رمزی نمودن داده خود در جلوگیری از تشخیص در تحلیل ایستا مورد کاربرد قرار می‌گیرند. با تغییر کلید رمزنگاری، کد بسته‌بندی‌شـــده بر اســـاس توزیع پیچیده ســـازی تحلیـــل انطباق الگو بازنویســـی می‌گردند.

دست‌کاری کد به‌صورت پویا: در این فن، توابع مشابه توســـط فراهم نمودن نمونه اصلی اصلاح‌شده قبل از اجرا مبهم‌سازی می‌شود.

نیازمندی‌های محیطی: در این راهکار، کلید رمزنگاری تنها به‌طور ایســـتا ذخیره نشـــده و فقـــط در هنگام ایجاب شرایطی خاص، تولید می‌شود. این راهکار به‌طور گسترده در حوزه ساخت بدافزار مورد کاربرد قرار می‌گیرد.

37- Stuxnet
38- Debug

جدول ۲: سناریوهای تحلیلی ایستا و پویا

بازنویسی پویا کد	بازنویسی ایستا کد	مبهم‌سازی داده
بسته‌بندی	گزاره مات	ضبط داده
دست‌کاری پویای کد	تزریق کد مرده	رمزگذاری
مبهم‌سازی کد به‌واسطه سخت‌افزار	درهم‌سازی اسامی	تبدیل داده ایستا به روال‌ها
مجازی‌سازی	مرتب‌سازی مجدد	درج گره مخرب
فنون ضد برگردانی و اشکال‌زدایی	جایگزینی دستورالعمل‌ها	کاوش
	بازترکیب و قطعه‌قطعه سازی توابع	
	حذف فراخوانی‌های کتابخانه‌ای	
	مبهم‌سازی جریان کنترلی	
	حذف وابستگی‌ها	

مبهم‌سازی کد به‌واسطه سخت‌افزار: نشانه[39] سخت‌افزاری جهت بهبود فنون دیگر مبهم‌سازی مورد کاربرد قرار می‌گیرند. ایده پایه، ساخت نرم افزار–سخت افزاری بوده که تولید نرم‌افزار را به‌واسطه حضور نشانه‌های سخت‌افزاری ایجاد می‌نماید. بدون این نشانه‌ها، تحلیل برنامه به علت عدم حضور اطلاعات مهم و موردنیاز، ممکن نخواهد گشت. سازوکار جداسازی مبتنی بر سخت‌افزار به نرم‌افزار اجازه ممانعت در برابر دسترسی به مکان بخصوصی از حافظه توسط سایر نرم‌افزارها و حتی هسته[40] سیستم‌عامل را صادر می‌کند.

مجازی‌سازی[41]: مجازی‌سازی بیانگر تبدیل عملکرد برنامه به بایت کد برای مفسرهای[42] پیش‌فرض ماشین مجازی بوده که به همراه برنامه قرار می‌گیرد.

فنون ضد اشکال‌زدایی و برگردان: این طبقه‌بندی از مبهم‌سازی دربرگیرنده راهکارهایی است که به‌واسطه استفاده از برگردان‌ها و یا اشکال‌زدایی، تلاش در تحلیل کد دارند.

۵- تحلیل کد

چهار رده تحلیلی ذکرشده برای مقابله با برنامه‌های محافظت‌شده به‌واسطه مبهم‌سازی از قبیل انطباق الگو، تحلیل ایستای خودکار، تحلیل پویای خودکار و مهندسی معکوس وجود دارد.

۵-۱. انطباق الگو

هدف از تطبیق الگو، اشاره به فنون خودکار برای کاوش حول ساختارهای شناخته‌شده در برنامه دودویی است که به‌واسطه ابزارهای مبهم‌سازی و یا شناسه‌های بخصوصی تحت عنوان کتابخانه و نمونه‌های کد، شناخته می‌شوند. در تحلیل‌های ایستا و پویا، تطبیق الگو صرفاً نحوی بوده و هیچ ارتباطی با برنامه معنایی ندارد. به‌عنوان‌مثال از تطبیق الگو می‌توان به تشخیص سریع کتابخانه و به رسمیت شناختن‌ها در اشکال‌زدای « آی دی اِی »[43] اشاره نمود که از شناسه‌ها جهت تشخیص توابع کتابخانه‌ای باهدف اعمال مهندسی معکوس استفاده می‌کند. شناسه‌های توابع یا همان کتابخانه‌ها در پایگاه داده‌ای نگه‌داری می‌شوند. مادامی‌که کد دودویی جدیدی برگردان شود، بایت‌های آن‌ها در برابر شناسه‌های شناخته‌شده بررسی می‌شوند. توابع به رسمیت شناخته‌شده، نام‌گذاری و نگه‌داری شده و سپس توسط عامل انسانی در عملیات مهندسی معکوس برای درک عملکرد کد، مورد کاربرد واقع می‌گردند. فن فوق‌الذکر، تحلیلی چند سطحی در مرحلهٔ‌های اولیه ایستا و پویا برای مثال در سطح گراف جریان کنترلی و یا ردیابی دستورالعمل‌ها ایجاد می‌نماید.

حالت پایه تطبیق الگو، محدود به مقایسه کدهای برنامه

39- Token
40- Kernel
41- Virtualization
42- Interpreter
43- IDA pro

در سطح بایت در مقابل بعضی الگوهای از پیش تعریف‌شده است. این نوع تطبیق الگو که دربرگیرنده تبدیلات مبهم‌سازی است در مقابل روش‌های دست‌کاری کد ضعیف است. اصولاً مفاهیم پیچیده تطبیق الگو اغلب برای شناسایی رفتارهای مخرب به‌کار می‌روند. بعضی از فنون پیشرفته انطباق الگو، توانایی مقابله با فنون مبهم‌سازی را در مرحلهٔ تحلیلی همچون دستورالعمل‌های هم‌ارز دارا هستند. دربندهای زیر، نسبت به معرفی تحلیل مبتنی بر تطبیق الگو در حضور رده‌های مختلف مبهم‌سازی اقدام گردیده است.

موقعیت‌یابی داده: فنون تطبیق الگو بر روی‌داده به‌محض تغییر در ساختار، مورد شکست واقع می‌گیرند؛ بنابراین، حتی ساده‌ترین نوع مبهم‌سازی در برابر حالات ساده تطبیق الگو، کارا است. فنون ضد برگردان کردن، در برابر تحلیل‌های دودویی تطبیق الگو نیاز به برگردانی برنامه ندارند.

موقعیت‌یابی کد: این سناریو به‌واسطه تطبیق الگو اغلب در متون پژوهشی مرتبط با بدافزارها ذکرشده است. هدف اصلی در این سناریوی تحلیلی، تولید الگوهای عمومی بوده که رفتارهای مخرب را تشریح می‌نماید. این فنون، هدفی در درک برنامه معنایی نداشته و همواره توانایی موقعیت‌یابی تولید کد با رفتار غیرعادی و مخرب رادارند. نرم‌افزار « هنکاک »[44] سیستمی برای تولید خودکار شناسه‌های بدافزاری، به‌واسطه نرمال‌سازی کدهای دستورالعمل در برابر تبدیلات ساده کد همچون بازچینش دستورالعمل‌ها است. در سویی دیگر، فنون پویای مبهم‌سازی کد همچون مجازی‌سازی، ساختار کد را به‌طورکلی دست‌کاری و حذف می‌نمایند. ازاین‌رو، تفسیر تطبیق الگو مبتنی بر راهکارهای تحلیلی کاملاً بی‌اثر خواهد بود. شایان‌ذکر است که فنون ضد برگردان، محافظت بخصوصی را در مواجهه با راهکارهای تطبیق الگو باهدف موقعیت‌یابی کد فراهم نخواهند نمود.

۵-۲. تحلیل ایستا

تحلیل ایستا به‌طور گسترده برای بهینه‌سازی کد، یافتن یا اثبات عدم وجود خطا، آسیب‌پذیری‌ها و مهندسی معکوس مورد کاربرد قرار می‌گیرد. در عمده‌ترین حالت ، تحلیل ایستا بدون اجرای متعدد برنامه دلخواه در حالت واقعی و یا ماشین مجازی، با تحلیلگری خاص به بررسی کدهای اجرایی و یا بازنمایی کد برگردان‌شده می‌پردازد. این تحلیل اطلاعات برنامه را به‌واسطه ارزیابی تعداد دفعات اجرای ممکن برنامه هدف، استنتاج می‌نماید. تحلیل ایستا در صورتی پایدار است که متضمن تمامی حالات اجرایی ممکن برنامه باشد. برای نمونه، تحلیل ایستا که تلاش در تولید گراف کنترل جریان از برنامه‌ای دودویی را دارد، در صورتی پایدار است که گراف نتیجه، شامل حداقل تمامی تبدیل‌های کنترلی ایجادشده در زمان اجرا برنامه باشد. به دلیل تعمیم‌ناپذیری، تحلیل ایستا تنها به‌واسطه تقریب بیش‌ازحد، پایدار خواهد بود. برای مثال، با تعمیم رفتار واقعی برنامه و پذیرش آن، این تعمیم دربرگیرنده رفتاری است که در زمان اجرای واقعی رخ نخواهد داد. گراف کنترل جریان پایدار نباید محتوی لبه‌های در نظر گرفته‌شده در زمان اجرا باشد. تقریب بیش‌ازحد در این تحلیل، عاملی در جهت کشف بدافزار بوده که هزینه زیادی را در بردارد. اغلب سیستم‌های کاربردی در تحلیل ایستا، به‌عنوان فنون ضعیف در نظر گرفته می‌شوند. از دیدگاه مفهومی، مبهم‌سازی دقت تحلیل ایستا را تنزل می‌بخشد. برای مثال، مبهم‌سازی، منابع مازادی را در حین ارزیابی اضافه می‌کند. در کنار نیازمندی‌های مفهومی، چالش‌های فراوانی برای تحلیل کد مبهم‌سازی شده وجود دارد. اغلب ابزارهای در دسترس، فرضیاتی پیرامون رفتار ساختار کد اجرایی گمارده که به‌واسطه مبهم‌سازی در هم نوردیده شده‌اند. اغلب تحلیلگرهای ایستا، حتی باوجود مبهم‌سازی در مرحلهٔ تحلیل بدافزار با شکست مواجه می‌شوند. این فرایند کاملاً مشابه با باج افزارهای چندریختی و فراریختی خودرمزکننده اخیر بوده که تحلیل آن بسیار دشوار است و در سویی دیگر بازه مخاطراتی آن شامل رمزی‌شدن داده‌های حساس کاربر و نیز از دست رفتن اطلاعات

44- Hancock

وی خواهـد بود. علیرغم این موضوع، تحلیل ایسـتای کد مبهم‌سـازی شده توانایی ارائه اطلاعات بخصوصی را در فرایند مهندسـی معکوس و درک عمیق‌تر از کد و همچنین اطلاعات اضافه‌ای در راسـتای مبهم زدایی در پی خواهد داشت.

در بازه‌ای متفاوت و مشابه، فنون تحلیل ایستا توانایی پایدارسـازی عملیات مبهم‌سـازی رادارند. در سناریوی تعیین موقعیت داده مشـابه تطبیق الگـو، تحلیل خودکار ایسـتا درزمانی که تحلیل داده در بازنمایی خود نگه‌داری نمی‌شود، بسیار دشوار است. فن بازنویسی پویای کد در برابر تعیین محـل داده درون برنامـه، به‌عنوان بازنمایی اصلی داده دست‌کاری شده، سودمند است. از طرفی دیگر، تحلیل ایسـتا به‌عنوان تنها کدی که مفسـر توانایی تحلیل مسـتقیم آن را دارد، به‌طور عمومی در برابر مبهم‌سازی مبتنی بر مجازی‌سازی اثربخش است. در تحلیل ایستا این فرایند تحت عنوان هم‌سطح سازی دامنه یادشده که در طی آن، اطلاعات جریان داده به محلی در مفسر ادغام گردیده کـه نتایج و تأثیرات خاص خود را در تحلیل مبهم‌سـازی خواهد داشت.

در تحلیل ایستا، تعیین وضعیت ویژگی‌های بخصوص برنامه در سـطح دودویی مبتنی بر تحلیل ساختار گراف کنترل جریان برنامه است. ازاین‌رو راهکارهای مبهم‌سازی کد همچون تبدیل حلقه، موازی‌سازی کد، گزاره‌های مات و ... که جریان کنترلی برنامه را دسـت‌کاری و پنهان‌سازی می‌نماینـد، به‌عنوان داوطلبی برای محافظت در برابر فنون تحلیل ایستا در نظر گرفته می‌شود. گزاره‌های مات، مفهوم بسـیار مهمی برای مبهم‌سـازی جریان کنترلی در برابر ابزارهای تحلیل ایستا است. شایان‌ذکر است که مبهم‌سازی کد مبتنی بر بازنویسـی پویا کد، تحلیل ایسـتا را بسـیار دشوار خواهد نمود.

۵-۳. تحلیل پویا

تحلیل پویا که امروزه به‌طور گسـترده به‌عنوان بخش مهمی از فنون کشـف جرم و تحلیل بدافزار به‌کار می‌رود، به بازبینی رفتار سیسـتم در حال اجرا می‌پردازد. به‌طور خاص، تمامـی نرم‌افزارهای تحلیلگر از راهکار تحلیل پویا در سازوکار خود اسـتفاده می‌نمایند. برای نمونه، تحلیل پویا نسـبت به ضبـط فراخوانی‌های سیسـتمی، اجرای دستورالعمل‌ها و یا ردیابی جریان کنترلی اقدام می‌کند. این تحلیل زیرمجموعه‌ای از تمامی اثرهای یک برنامه را داشته و تحت تقریب عمـل می‌نماید. مزیـت راهکارهای پویا به نسبت ایستا در تحلیل کد، توانایی سهولت اعمال در حوزه دودویی اسـت. در حقیقت، تحلیل کد دودویی نسبت به کد منبع آسان‌تر اسـت زیرا ذخیره ردیابی‌های در حال اجرا بیانگر نشانی دستورالعمل‌ها در سطح دودویی و نه‌تنها در سطح کد منبع برنامه است. این در حالی است که فنون ضد اشکال‌زدایی، توانایی مخالفت با تلاش‌های تحلیلی رادارند. در تحلیـل کد، راهکارهای پویا به نسـبت ایسـتا توانایی سـهولت اعمال در حوزه دودویی را دارا هسـتند. تحلیل پویا آنچه را کـه در زمان اجرا در جریان اسـت بازبینی می‌کند. ازاین‌رو مبهم‌سازی به‌تنهایی توانایی پنهان‌سازی رفتار برنامه را بر عهده نخواهد داشت. فنون شناخته‌شده مبهم‌سازی بجز رمزنگاری (جعبه سفید) کاربرد محدودی در سناریوهای تحلیل پویا دارد که طی آن داده در طی زمان و همچنین در بعضی از نقاط قابل‌مشاهده است. در تحلیل پویا، اغلب فنون تحلیل ایستا همچون بازنویسی پویای کد، امنیـت بخصوصی را در محافظـت از داده اعمال نمی‌کند و تنها بخش، مبهم‌سـازی بوده که مسـتلزم زمان اجرای بخصوصی برای اجرا اسـت. فنونی از قبیل نیازمندی‌های محیطی و کدهای وابسـته به سخت‌افزار، توانایی مقاومت در برابر تحلیل‌های پویا را در شرایطی خاص همچون عدم وجود زمان اجرا دارند.

تعیین محل قرارگیری کد: تعیین ویژگی‌های بخصوص در کد دودویی به‌واسطه تحلیل پویای مبتنی بر مشاهدات رفتـاری برنامـه امکان‌پذیر اسـت. فنون بازنویسـی کد ایسـتا همچون جایگذاری دسـتورالعمل‌ها، درج کد مرده، گزاره‌های مات، درج کد هرز در سناریوهای تحلیلی پویا به

جدول ۳: بررسی فنون مبهم سازی [۲]

		تحلیل به‌واسطه عامل انسانی				تحلیل پویا خودکار				تحلیل ایستا خودکار				تطبیق الگو	
مبهم‌سازی	نام	UC	EC	LC	LD	UC	EC	LC	LD	UC	EC	LC	LD	LC	LD
مبهم‌سازی داده	مرتب‌سازی داده								✓				✓		
	تغییر رمزگذاری								✓						✓
	تبدیل داده‌های ایستا به روال				✓				✓						✓
بازنویسی ایستا کد	جایگذاری دستورالعمل‌ها											✓		✓	
	گزاره مات											✓			
	درج کد مرده					✓									
	درج کد نامربوط													✓	
	بازترکیب														
	تبدیل حلقه														
	بازترکیب/ قطعه‌قطعه سازی توابع					✓									
	عدم تعیین پیش‌فرض					✓									
	مبهم‌سازی جریان کنترلی	✓		✓											✓
	موازی‌سازی کد														
	درهم‌سازی اسامی														
	حذف فراخوانی‌های استاندارد کتابخانه‌ای	✓													
	حذف وابستگی‌ها														
بازنویسی پویا کد	بسته‌بندی	✓		✓				✓				✓		✓	
	دست‌کاری پویا کد														
	نیازمندی‌های محیطی														
	مبهم‌سازی کد به‌واسطه سخت‌افزار														
	مجازی‌سازی	✓		✓				✓				✓	✓		
	فنون ضد برگردان	✓				✓		✓		✓					

دلیل این‌که اغلب آن‌ها صرفاً برای ممانعت از تحلیل کد پویا در نظر گرفته‌شده‌اند، سودمند نیستند.

استخراج پویای کد، مستلزم برقراری ارتباط بین وابستگی‌های اجزای مختلف برنامه است. رده‌های مبهم‌سازی ایستای متعددی برای دشوارسازی فرایند تحلیل وابستگی‌ها وجود دارد. همواره فهم کد مستلزم درک عمیقی از اصول پایه مبهم‌سازی در تحلیل‌های خودکار است.

۴-۵. مهندسی معکوس و تحلیل به‌واسطه انسان

عملیات مهندسی معکوس به‌واسطه عامل انسانی و فنون خودکار شناسایی اسامی و روابط بین رده‌ها به نسبت سایر تحلیل‌ها بسیار راحت‌تر خواهد بود. از طرفی دیگر، شناسایی توانایی‌های عامل انسانی بسیار دشوار است. اغلب فنون مبهم‌سازی، بازه محدودی در سناریوهای مهندسی معکوس داشته که عامل انسانی تلاش در تعیین ساختار داده آن برنامه را دارد. یکی از مفاهیم مهم در محافظت از داده درون کد دودویی، رمزنگاری جعبه سفید است. این فن برای ممانعت از استخراج کلید رمزنگاری به‌واسطه ترکیب آن با الگوریتم در قالب دودویی، مهیا می‌شود. اگرچه در دهه‌های گذشته الگوریتم‌های رمزنگاری جعبه سفید استاندارد رمزنگاری

پیشـــرفته[45] و اســـتاندارد رمزنگاری داده[46] مطرح‌شده بود ولیکن این فن‌ها به‌واسطه راهکارهایی همچون تحلیل آماری، نفوذ با خطا، پیاده‌ســـازی فشـــرده، رمزنگاری تفاضلی و تحلیل رمـــز، مورد نفوذ واقع گردیده‌اند [۲]. بعضی از فنون مبهم‌ســـازی، انتزاع کد را حذف می‌کنند. درهم‌ســـازی نام، حـــذف فراخوانی‌هـــای کتابخانه‌ای اســـتاندارد و نیز حذف وابستگی‌ها توانایی سخت سازی مهندسی معکوس رادارند. یکی از فنون در حوزه استخراج کد به‌واسطه عامل انسانی، تکه‌تکه‌سازی[47] اســـت. این راهکار مبتنی بر ایده کاهش کد، جهت کمینه‌سازی تکه‌هایی اســـت که رفتار بخصوصی از برنامه را تعریف می‌نمایند. جدول ۳ بیانگر مفاهیم بیان‌شده در این نوشـــتار است. در جدول مذکور، پارامترهای LD به محـــل داده، LC بـــه محل کد، EC به اســـتخراج کد و UC به درک کد اشاره می‌نماید. شایان‌ذکر است که بندهای مشکی بیانگر شکست اساســـی تحلیل به‌واسطه مبهم‌سازی است. در بخش‌های خاکستری، مبهم‌سازی قابل شکست نبوده و تحلیل را بســـیار گران عرضه می‌دارد. در بخش‌های سفید، مبهم‌ســـازی به‌صورت جزئی در فرایند تحلیل هزینه به بار می‌آورد. بخش‌هایی که با علامت صحیح مشخص‌شده‌اند، دربرگیرنده پشتیبانی نرخ مذکور در پژوهش‌های انجام‌شده اســـت. در خاتمه گزینه‌هایی که بدون علامت صحیح در کل شـــکل درج‌شـــده‌اند، صرفاً تحت عنوان ارزیابی‌های نظری عرضه‌شده‌اند.

۶- نتیجه

با افزایش نفوذ و ربایـــش نرم‌افزار، تلاش نوینی برای بحـــث و پیاده‌ســـازی فنـــون کلی مبهم‌ســـازی به‌صورت مفهومی در این نوشـــتار انجام گردیده است. به‌طور عادی بعد از مبهم‌سازی، پیچیدگی کد بر اساس منطق و ساختار به دلیل درج تصادفی، دســـت‌کاری، برداشـــت و بازآرایی افزایش می‌یابد. اگرچه تمامی روش‌های مبتنی بر نرم‌افزار توانایی حفاظت کامل از اطلاعات را ندارند، ولیکن با ادغام تحلیل‌های ایســـتا و پویا و نیز گســـتره اهداف تحلیلگر که به‌طور مجزا در این نوشـــتار مطرح گردیده اســـت، به‌طور عمده‌ای کار را برای مهندسی معکوس و همچنین تشخیص مبتنی بر شناسه و تشخیص کد از میان تطبیق الگو دشوار می‌ســـازد. نتایج بیانگر این اســـت که مقاومت مبهم‌سازی به‌شـــدت به اهداف تحلیلگر و منابع در دســـترس، وابسته است. به‌عنوان نتیجه، محافظت از مالکیت معنوی در برابر عامل انســـانی همچنان به‌عنوان چالشـــی بزرگ به‌حساب می‌آیـــد. هدف اصلی در پژوهش‌هـــای آتی، کنترل رفتار و کشف آگاهانه معنایی بدافزارهای چندریختی و فراریختی است.

45- Advanced Encryption Standard (AES)
46- Data Encryption Standard (DES)
47- Fragmentation

مراجع

[۱] اکبری‌فر، امیر، مرتضوی، رضا، «راهکارهای مبهم‌سازی در جهت محافظت از کد»، دومین همایش ملی علوم محاسباتی، ۱، ۷، دامغان، ۱۳۹۵.

[2] Schrittwieser, S., Katzenbeisser, S., Kinder, J., Merzdovnik, G., & Weippl, E. (2016). Protecting Software through Obfuscation: Can It Keep Pace with Progress in Code Analysis? ACM Computing Surveys (CSUR), 49(1), 4.

[3] Christodorescu M, Jha S. Testing malware detectors. ACM SIGSOFT Softw Eng Notes 2004;29(4):34e44.

[4] Christodorescu M, Jha S, Seshia S, Song D, Bryant R. Semanticsaware malware detection. In: Security and privacy, 2005 IEEE symposium on. IEEE; 2005. pp. 32e46.

[5] Wu Z, Gianvecchio S, Xie M, Wang H. Mimimorphism: a new approach to binary code obfuscation. In: Proceedings of the 17th ACM conference on computer and communications security. ACM; 2010. pp. 536e46.

[6] Lyda R, Hamrock J. Using entropy analysis to find encrypted and packed malware. Secure Priv IEEE 2007;5(2):40e5.

[7] Bilar, D. (2007). Opcodes as predictor for malware. International Journal of Electronic Security and Digital Forensics, 1(2), 156-168. doi:10.1504/ijesdf.2007.016865

[8] Damodaran, A., Troia, F. D., Visaggio, C. A., Austin, T. H., & Stamp, M. (2017). A comparison of static, dynamic, and hybrid analysis for malware detection. Journal of Computer Virology and Hacking Techniques, 13(1), 1-12. doi:10.1007/s11416-015-0261-z

[9] Rana, H., & Stamp, M. (2014). Hunting for Pirated Software Using Metamorphic Analysis. Information Security Journal: A Global Perspective, 23(3), 68-85. doi:10.1080/19393555.2014.975557

[10] Lin, D., & Stamp, M. (2011). Hunting for undetectable metamorphic viruses. Journal in Computer Virology, 7(3), 201-214. doi:10.1007/s11416-010-0148-y