

# ParsBook.Org

پارس بوک، بزرگترین کتابخانه الکترونیکی فارسی زبان

# ParsBook.Org



The Best Persian Book library

❖ Debugger

❖ Disassembler

❖ Decompiler

## Debugger

در ادبیات مرسوم مباحث مرتبط با امنیت نرم افزار ، دیباگر ، نرم افزاری است که امکان بررسی دقیق روند اجرای یک برنامه اجرائی یا کتابخانه یا درایور را فراهم میکند و در این روند ، کنترل کامل و جزئی حافظه ، متغیرها و توابع ، تراکنشهای پردازنده و حافظه و دیسک و ... و ارتباط با کتابخانه های اشتراکی در اختیار دیباگر میباشد .

دیباگر برنامه اجرائی یا کتابخانه یا درایور را با ترتیباتی قابل پیکره بندی فراخوانی میکند ، به توسعه گر یا آزمایش کننده این امکان را میدهد که در مواقع به خصوصی روند اجرا را متوقف یا منوط به وقوع اتفاق خاصی نماید ، یا حافظه را برای کشف مقدار یا مقادیر به خصوصی جستجو کند یا تراکنشهای برنامه با کتابخانه های همراه ، کتابخانه های سیستم عامل ، سخت افزارهای همراه و... را کنترل نماید . در واقع وظیفه اصلی یک دیباگر کنترل کامل فرآیند اجرا شدن یا فراخوانی شدن یک برنامه یا کتابخانه اشتراکی است . اطلاعاتی که میتوان با استفاده از دیباگرها بصورت معمول بدست آورد:

- هر آنچه بین نرم افزار و حافظه اتفاق می افتد . آدرس دهی متغیرها و توابع ، جایگزینی مقادیر ، تخصیص حافظه و آزاد سازی ، محل دستورات زبان ماشین در حافظه ، داده های در حال ارسال به پردازنده و...
- وضعیت لحظه به لحظه پردازنده ؛ محتویات رجیسترها و...

- هر آنچه بین نرم افزار و کتابخانه هائی که از آن تغذیه میکند واقع میگردد . فراخوانی توابع ، کلاسها یا اشیاء ؛ آدرس کتابخانه ها و توابع مربوطه در حافظه و...

عموم دیباگرها متداول غیر از موارد فوق اطلاعات متنوع دیگری را نیز فراهم می کنند که بسته به مورد و نوع کاربرد متفاوت هستند .

انواع دیباگر ها : از دید لایه کاربرد:

### ApplicationMode

### Kernel Mode

دیباگرهای Application Mode فقط به Ring3 و فضای User Mode سیستم عامل دسترسی دارند و امکان Resolve آدرسها فقط در این محدوده برایشان موجود است اما دیباگرهای Kernel Mode یا اصطلاحا Kernel Debugger ها امکان دسترسی به فضای آدرسی Ring0 یا Kernel Space را نیز دارند و در صورت وجود Symbol file های ، امکان Resolve آدرسها در فضای کرنل نیز برایشان وجود دارد .

از دید نوع عملکرد:

### • Source code Level

### • Assembler Level

دیباگرهای Source Code Level با استفاده از سورس کد برنامه ، وظایف مذکور رو کنترل و بررسی میکنند و عموماً هنگام توسعه ی یک نرم افزار ، به خطایابی یا ایجاد روندهای کنترل خطای منطقی کمک بسیاری می کنند اما دیباگرهای Assembler Level بدون استفاده

یا نیاز به سورس کد و صرفاً با داشتن نسخه‌ی باینری، وظایفشون رو انجام میدن. امکان کنترل کد، در سطح Assembler وجود دارد یعنی فرضاً می‌تونید روی دستوری مانند Call یا Test یا Mov برنامه مورد نظر یک BreakPoint بگذارید.

دیباگرها اولین وسیله مورد استفاده نفوذگران نرم افزاری است فلذا روتین‌های آنتی دیباگ همیشه یکی از عناوین مطرح در روشها و ابزارهای حفاظت از نرم افزار هستند. این روتین‌ها عموماً از این روشها برای شناسایی دیباگر استفاده می‌کنند:

❖ استفاده از مشخصات فردی دیباگر: نام پروسه، مشخصات ثبت شده در رجیستری، مولفه‌های هدر و...

❖ بررسی حافظه مجازی اختصاص یافته به برنامه و تست وجود دیباگر  
Hook آدرس یا Hook آدرس آدرس توابع دیباگ ویندوز و بررسی وجود پوینتر  
به آنها یا عدم وجود آن

❖ بررسی پردازنده (مخصوص روتین‌های Ring0)

برای تمامی این Trick ها و روشهایی که از هر کدام منشعب میشود، نیز، بالتبع روشهای متقابلی وجود دارد؛ در مجموع چیزی به عنوان متوقف کردن قطعی همه دیباگرها منطقی نیست؛ بهترین روتینهای موجود که هر کدام با مدتها تحقیق و بررسی ارائه شده اند در کمترین زمان ممکن by Pass شدن. استفاده از روتین‌های آنتی دیباگ برای حفاظت از نرم افزار روش مناسبی برای کندتر کردن فعالیت نفوذگران تازه کارتر است، نه چیزی بیشتر.

بجای لیست کردن دیباگرها و توضیحات غیر مفید در مورد هر کدام، توصیه‌های شخصی

ام‌رو مینویسم که صرفاً نشان دهنده‌ی دیدگاه خودم است نه چیز دیگر.

## Windbg

دیباجر مایکروسافت . رابط کاربری نسبتا خوبی داره و به خوبی با Windows Symbol files متصل میشه . مطمئنا داشتنش برای کسانی که به هر دلیل به دیباگر نیاز دارند یک ضرورته

## Ollydbg

قدرتمند ترین دیباگر Assembler Level و Application Mode موجود. اسکریپتهای متعددی برای خودکار سازی بسیاری از وظایف توسط افراد مختلف براش نوشته شده . زندگی بدون Olly برای نفوذگران نرم افزاری قطعا غیر ممکنه . Olly توسط BCB6 توسعه داده شده/میشه .

## SoftICE

قدرتمند ترین دیباگر Kernel Mode که بصورت همزمان Source Level و Assembler Level نیز هست . بجای تکیه بر رابط کاربری ، دستورات پیچیده ای داره که انعطاف بیشتری به روند دیباگ میده . غیر حرفه ای ها نمیتونن رابطه خوبی باهاش برقرار کنن . نسخ متعددی داره که هر کدوم رو باید تو یه فضای خاص استفاده کرد . بعدا ممکنه یه تاپیک مجزا براش ایجاد بشه Visual SoftICE . به عنوان آخرین نسخه ارائه شده ، همراه با DriverStudio ی DevPartnet ارائه میشه و امکانات بصری خوبی به نسخه های سنتی SICE اضافه کرده .

## : kd

یا Kernel Debugger ، دیباگر استاندارد DDK ویندوز یا Driver development Kit است . یک Kernel Mode درایور که هم بصورت Assembler Level و هم بصورت

Source Level کار میکند. با SICE قابل مقایسه نیست اما چون سازگاری خوبی با Windows Symbol Files داره ، برای دیباگ درایورها فوق العاده مفیده .

دیباگرهای دیگری هم وجود دارن که هر کدوم ممکنه نسبت به موارد فوق مزایا یا نقائصی داشته باشن ؛ لیکن تجربه نشون میده OllyDBG برای اغلب کاربردها ، SI برای حرفه ای ها و kd نهایتا برای توسعه گران Ring0 ، همه نیازها رو برطرف میکنند . این مطلب در ادامه به معرفی ویژگیهای Disassembler ها و Decompiler ها خواهد پرداخت . این مطلب قراره الفبای این حوزه رو به کسانی که باهاش غریبه هستن یاد بده ، نه چیزی بیشتر .

## Disassembler

ابزار برای تبدیل کدهای زبان ماشین ، به معادل اسمبلی آنهاست . زبان اسمبلی به عنوان یک زبان سطح پائین ، به ازای یک دستور یا اصطلاحا Instruction به یک دستور زبان ماشین تبدیل خواهد شد فلذا بازگردوندن کد باینری به معادل اسمبلی اون کار چنان دشواری نیست .

**سوال :** یک برنامه اجرایی PE تحت ویندوز داریم که با MASM یا Macro Assembler نوشته شده است . با یک Disassembler آن را به کدهای اسمبلی تبدیل میکنیم . آیا خروجی لزوما با کد اسمبلی اولیه یکسان خواهد بود ؟

**جواب :** لزوما خیر Disassembler . های امروزی عموما به اندازه کافی هوشمند و خوب هستند اما همیشه انتظار داشت هوش داشته باشند . این ابزارها با استفاده از Knowledge Base ای که برای کامپایلرهای مختلف داخلشون تعبیه شده ، کدهای باینری رو به معادل اسمبلی تبدیل میکنند اما تضمینی وجود نداره توسعه گر اصلی برنامه ، نیز دقیقا همین کدها رو نوشته باشه .

میزان خطای Disassembler ها رابطه مستقیمی با میزان پیچیدگی و سطح بالا بودن زبان داره . یعنی Disassembler با دریافت ورودی که با MASM یا GCC تولید شده خطای کمتری خواهد داشت نسبت به زمانی که قراره فایل باینری تولید شده توسط دلفی یا VC رو بررسی کنه ؛ خصوصا اگر کتابخانه های مفصلی مثل VCL و MFC هم استفاده شده باشه ؛ لیکن در مجموع همیشه تا حدود زیادی به خروجی Disassembler های امروزی برای درک منطق و روند فعالیت برنامه اعتماد کرد.

سوال Disassembler : چه کاربردهائی دارد ؟

جواب : فرض کنید قرار روتین مقایسهء سریال نامبر یک Protection کودکانه رو بررسی کنید . دیباگر بهتون کمک خواهد کرد تا بتونید نقل و انتقال مقادیر بین متغیرها و محل انجام مقایسه رو پیدا کنید ؛ و Disassembler کمک خواهد کرد با مشاهدهء دقیق و تک تک دستورات ، انتخابهای خوبی برای عبور از اون حفاظ یا تولید یک Patch و دستکاری نسخهء باینری برنامه داشته باشید . برای یک نفوذگر نرم افزاری حرفه ای ، مطالعهء خروجی Disassembler از یک روال حفاظتی ، معادل مطالعهء سورس کده !

Disassembler های معتبر و قابل اتکاء تحت ویندوز یعنی IDA Pro و WinDasm و

PView، دارای امکانات دیگری هم هستند :

-آنالیز کد و ایجاد ارتباطات بصری بین اجزاء و روتینهای مختلف برنامه ؛ نمایش توالی

اجرا توابع و کاربری از اشیاء و...

-آنالیز کد باینری و تشخیص توابع داخلی بکار برده شده توسط کامپایلرها و ارائه

کامنت های مفید

## ارائه کردن امکانات یک دیباگر در کنار Disassembler

...و

در مجموع ، Disassembler به عنوان دومین ابزار مهم در حوزه امنیت نرم افزار ، یکی از عناصر لا ینفک یک روند حرفه ای آنالیز و بررسی باینری است IDA Pro . توسط یک تیم توسعه گر هماهنگ و قدرتمند قدرتمند ترین Disassembler موجود در محیط ویندوز است . ( یک نسخه مبتنی بر لینوکس هم داره ، لیکن انتظارات نسخه ویندوزی رو همیشه ازش داشت ) این محصول با Borland C توسعه داده میشه و یک دوره فشرده آموزشی اون توسط خود شرکت توسعه گر ، به مدت چهار روز ، برای هر نفر ، چیزی حدود هزار و پانصد دلار هزینه در بر خواهد داشت WinDasm . دیگه تحت توسعه نیست و نسخه های جدیدتری نخواهد داشت .

آخرین نسخه رسمی ۸,۹ است که دو نسخه ۹ و ۱۰ هم توسط افرادی که براش Patch هائی نوشتن منتشر شدن WinDasm . مدتها به عنوان ابزار شماره یک استفاده شده ( خصوصا با توجه به گران قیمت بودن IDA ) و بسیاری از مقالات و راهنماهای موجود مبتنی بر اون نوشته شده اند IDA Pro . با داشتن آنالایزر قدرتمند و امکان شناسائی کتابخانه های مختلف ، داشتن پلاگینها متعدد و قابلیت های بی شمار ( که بی اغراق امکان نداره حتی بشه در غالب یک کتاب در مورد همشون حرف زد ) بهترین گزینه موجوده هر چند تسلط به اون حتی برای کسانیکه دانش بالائی دارند واقعا دشوار و پر هزینه خواهد بود .

## Decompiler

یک کتابخانه یا درایور به سورس کد اصلی ؛ قبل از ورود به بحث لازمه یک طبقه بندی از

موجودیتهائی که ممکنه ذیل عنوان Decompiler مطرح بشن داشته باشیم:



- برنامه های اجرایی باینری : برنامه هائی که عموماً با زبانهای سطح بالائی نظیر VC یا دلفی نوشته میشن و به کدهای مخصوص به ویندوز/معماری ماشین ( مثلاً Win32/IA32 یعنی ویندوز ۳۲ بیتی روی اینتل ۳۲ بیتی ) ترجمه میشن .

- کتابخانه های اشتراکی : بسته های نرم افزاری که عموماً با زبانهای سطح بالا برای کاربری در سایر برنامه ها تولید میشن و وابسته به سیستم عامل و معماری سخت افزاری هستن .

- برنامه های تفسیری : برنامه هائی که قبل از هر بار اجرا باید توسط یک مفسیر ترجمه بشن . به عنوان مثال برنامه های VB6 که بصورت PCode منتشر میشن و هر بار قبل از اجرا توسط VB runtime تفسیر میشن .

- برنامه های مبتنی بر زمان اجرا : برنامه هائی که برای اجرا نیاز به بستر از پیش فراهم شده ای برای روند اجرا دارند . مانند برنامه های دات نت و جاوا .

- درایور ها : کدهای سطح کرنلی که مختص سیستم عامل و معماری سخت افزاری هستن و عموماً با زبانهای سطح پائین تولید میشن .

**سوال :** آیا معنای تئوریک Decompiler برای همه این گروهها محقق شده؟ همیشه؟ خواهد شد؟  
**جواب :** خیر .

هیچ Decompiler ای برای گروههای اول و دوم و پنجم ارائه نشده ، همیشه ، نخواهد شد . یعنی دریافت سورس کد کامل نرم افزارهای اجرایی از نسخه باینری اونها مطلقاً غیر ممکنه . این عدم

امکان فنی نیست که در آینده با پیشرفت دانش امکان پذیر بشه ؛ یک نفی منطقی است . یعنی منطقاً امکان باز-تولید سورس کد کامل یک برنامه تولید شده با محیطهائی مثل Delphi یا VC وجود نداشته ، نداره ، نخواهد داشت .

**سوال :** پس نرم افزارهای متعددی که تحت عنوان Decompiler منتشر میشن چی ؟

**جواب :** با توجه به تعریف Decompiler ، جواب داده شد .

**سوال :** در مورد گروه های سوم و چهارم چی ؟

**جواب :** برای این دو گروه Decompiler وجود داشته و داره ؛ با یک توضیح کوچک . برنامه هائی هستند که میتونن از برنامهء اجرایی VB (به عنوان نماینده گروه سوم) یک سورس کامل قابل کامپایل تولید کنند ، اما ، این سورس ، لزوماً قرار نیست همان سورسی باشد که توسعه گران نرم افزار تولید کرده اند ؛ برای دات نت ( نمایندهء گروه چهارم ) نیز Decompiler های متعددی وجود داره ؛ اما هیچکدام قول نمیدهند خروجی آنها لزوماً همان سورس کدی باشد که برنامه از آن تولید شده .

**سوال :** آیا اصولاً وجود Decompiler لازمه ؟

**جواب :** برای اهداف مثبت و خیرخواهانه خیر . حتی برای اهداف غیر خیرخواهانه نیز وجود Decompiler یک لازمه نیست . هیچ کسی از وجود ابزاری که بتونه برنامهء او رو به سورس قابل قبولی مبدل کنه خوشحال نخواهد شد ؛ این ابزار کمکی به توسعه نرم افزار نمیکنه و سود اقتصادی ، پیشرفت علمی و افزایش قابلیتهای صنعت نرم افزار رو بیشتر نخواهد کرد . حتی برای اهداف مخرب هم ، وجود چنین ابزاری لازم نیست چون بسیاری از کسانی که در این مسیر فعالیت میکنند برای تخریب امنیت یک نرم افزار نیازی به دست رسی به سورس اون ندارند . کشف نقاط ضعف امنیتی یا عبور از حفاظتهای نرم افزار عموماً در محیطهائی اتفاق می افته که سورس وجود نداره و تمام فرآیند تخریب از طریق مهندسی معکوس یا Reverse Engineering انجام می

گیرد.

سوال : برنامه هائی که با جاوا و دات نت نوشته میشن چقدر امن هستند ؟

جواب : چون نقطهء صفری وجود نداره ، میزانی قابل ارائه نیست ؛ اما در مقام مقایسه :

- بررسی و Trace و بازبینی روند اجرای برنامه هائی که با محیطهائی نظیر دات نت و جاوا تولید میشوند ، به مراتب دشوار تر از برنامه هائی است که با محیطهائی نظیر دلفی و VC تولید میشوند؛ چرا که وجود Runtime های بزرگی مانند JRE یا CLR باعث میشه پیچیدگی فراخوانی ها ، مدیریت حافظه ، مدیریت ریسمان ها و پردازش ها و غیرهم به مراتب از برنامه های اصطلاحاً Native (مانند خروجی های VC) بیشتر باشه . پس فی المثل درک جزئیات فنی یک الگوریتم ، وقتی با دات نت نوشته شده باشه و به خوبی با Framework مخلوط باشه واقعا دشوار تر از درک جزئیات فنی الگوریتمی که با Delphi کامپایل شده.

-عبور یا تخریب حفاظهای نرم افزارهائی که با زبانهای نظیر جاوا و دات نت نوشته میشن به مراتب آسون تر از برنامه هائی است که با امثال دلفی و VC تولید میشن . چرا که اگر از درک جزئیات یک الگوریتم بگذریم ، وابستگی کامل این برنامه ها به یک لایهء میانی به نام زمان اجرا و عدم وابستگی به عناصر زیر ساختی سستم عامل و پردازنده و سخت افزار و همچنین امکانات بیشتر نفوذگران نرم افزار در تغییر محتویات این برنامه ها باعث میشن از این دیدگاه ، برنامه های Native وضع بهتری داشته باشند.

میگذریم از این حقیقت که برای یک حرفه ای ، اهمیت خاصی نداره که یک برنامه با دلفی

کامپایل شده یا Managed CPP

سوال : روشهای حفاظتی که برای مقابله با Decompiler ها مورد استفاده قرار می گیره چقدر قابل اعتمادند ؟

جواب : برای امثال دات نت و جاوا ، تقریبا هیچ . برای سایر محیطها ، Decompiler

دشمن خطرناکی به حساب نمیاد . فی المثل برنامه ای با عنوان DeDe با Delphi Decompiler مدعی است که یک Decompiler برای دلفی است ؛ اما در واقع تو فقط میتونی یک سری اطلاعات دریافت کنی ؛ و نه سورس کد کامل . ممکنه در برخی موارد این اطلاعات بتونه به یک نفوذگر نرم افزاری کمک خاصی بکنه ؛ اما من حیث مجموع ، اینگونه برنامه ها تهدید خطرناکی به حساب نمیان . بگذریم از این واقعیت که یک نفوذگر نرم افزاری برای حذف روتین حفاظتی نرم افزار یا جستجوی یک سرویس برای نقاط ضعف متداول ، نیازی به یک Decompiler نداره .

تمام وقایع تلخی که سالهاست شاهدش هستیم داره تحت شرایطی می افته که هیچ

Decompiler ضعیفی هم وجود نداره!

## Compressor

یا فشرده ساز ، ابزاری برای کاهش حجم فایل‌های [عموما] اجرایی است که کمک می کنه سایز و اندازه فایل ، روی دیسک یا اصطلاحا Disk Image کوچکتر بمونه ؛ کمپرسور مشخصا یک ابزار امنیتی نیست . اغلب توسعه گران از کمپرسورها برای کاهش حجم فایل‌های اجرایی غول پیکر برنامه هایشان استفاده میکنند ( برنامه هائی که هفتاد و پنج تا فرم و پنجاه تا کانکشن استرینگ و هزار و یک عدد ریسورس رو یکجا جمع کرده و فرصت نفس کشیدن به Loader بدبخت و مفلوک سیستم عامل نمیده و هر از چند گاهی اگر یک Access Violation هم به

برنامه نویس داده شده ، ایشون یک عدد بیلاخ حوالهء کامپایلر و محیط توسعه میکنند و توی فرومها و وبلاگها و غیرهم در نکوهش اون کامپایلر و محیط چه ها که نمی نویسند ! )

کمپرسورها منطق ساده ای دارند . یک برنامهء فوق العاده کوچک که برنامهء اصلی رو [ که قبلا فشرده شده و فرضا بصورت یک ریپسورس به اون لینک شده ] مجددا به یک فایل باینری/اجرائی روی حافظه مبدل کنه و شاید در نگاه اول بسیاری از مشخصه های برنامه اصلی رو به نوعی از دید بررسی کنندگان مخفی کنه ؛ اما در واقع ، آنچه اجرا میشه ، نسخهء کامل برنامهء اصلی ، روی حافظه است ؛ شاید کمپرسور بتونه یک برنامهء حجیم و خفن رو کوچکتر کنه و کمی از هارد دیسک چهل گیگابایتی شما رو نجات بده ، اما یقینا برنامهء بزرگ و خفن ، کماکان خرخرهء حافظه رو خواهد جوید و عموم کمپرسورها کمک قابل ذکری در این زمینه نمیکند ؛

بسیاری از توسعه گران مبتدی از کمپرسورها به عنوان یک ابزار حفاظتی استفاده میکنند به

این خیالات واهی :

- کد قابل مطالعه نیست ؛ چون حالا من روی دیسک چیزی از کد اصلی نمیبینم.
- چیزی از برنامه اصلی روی دیسک کپی نمیشه پس کسی نمیتونه برنامه رو دستکاری کنه.
- نوع کامپایلر و کتابخانه های مورد استفاده ام لو نمیره.

اما در واقع :

- با یک Memory Dump ساده ، کد باینری برنامهء اصلی به دست میاد و میشه

براحتی باهش PE ی اصلی رو بازسازی کرد.

- میشه متن کد برنامه رو روی حافظه دستکاری کرد و نهایتا برای تثبیت اون از

یک Loader استفاده کرد

- با مطالعه نسخهء روی حافظهء برنامه هم میشه براحتی عناصر برنامه رو آنالیز کرد

کمپرسورها، نه کمک مفیدی به حفاظت از امنیت نرم افزار میکنند، نه فایده چندانی به حال برنامه های عصر حجری حجیم و Single Module خواهند داشت؛ برنامه نویسهای با تجربه، برای رفع مشکل حجم کد، از تقسیم کد به عناصر کوچکتر، و برای حفاظت از امنیت کد از Packerها و Encryptorها استفاده میکنند، که به مرور در مورد نحوه عملکرد انواع مختلف و متداول آنها مطالبی ارائه خواهد شد

## Packer

Packer پرکاربردترین عنصر امنیتی در حوزه نرم افزار است. وظیفه Packer:

- تغییر شکل و قالب بخشهای مهم فایل های اجرایی
- ارائه روتینهایی برای حفاظت از برنامه روی حافظه
- کنترل یکپارچگی باینری
- مخفی نگهداشتن مشخصه های حائز اهمیت باینری مانند کامپایلر و کتابخانه ها و ... است. شاید بشه براحتی گفت بخش اصلی و پر رنگ روند نفوذ به حریم امنیتی یک نرم افزار، در UnPack کردن اون خلاصه میشه.

Packerها عموماً از این روشهای متداول برای انجام وظایفشان استفاده می کنند:

- ارائه روتین های آنتی دیباگ ( کشف دیباگر و توقف اون یا توقف روند اجرا برنامه یا ... )
- ارائه روتین های آنتی دامپ ( جلوگیری از بازسازی فایل باینری از طریق Memory Image )

- حفظ یکپارچگی فایل با تکنیکهایی مانند CRC و Hash و امضاهای دیجیتال

- رمزنگاری اجزاء مهم نرم افزار روی دیسک و حافظه و اجرا از طریق توابع درون ساخته نسبتاً

## امن Packer

-درهم ریزی ساختار قابل شناسایی باینری برای جلوگیری از باز-شناسایی ابزارهای توسعه

اغلب Packer ها بخش قابل توجهی از این نیازها رو پوشش میدن و برخی از اونها با ارائه روشهای ابتکاری امکانات دیگری هم در اختیار توسعه گر قرار میدهند ، مثلا حفاظت از کدهای سطح کاربر از طریق روتین های سطح کرنل ، یا اجرای کد رمز شده به کمک یک VM ( ماشین مجازی درون ساختهء Packer ) و ...

Packer های رایگان و تجاری متعدد و متفاوتی این روزها در دسترس هستند که هر کدام نقاط ضعف و نقاط قوتی دارند و از این مهم تر : برای اغلب اونها ابزارهای خودکار UnPack نیز موجود است . یعنی به محض استفاده از یک Packer متداول و شناخته شده ، یک نفوذگر نرم افزاری با تجربه براحتی اون رو میشناسه و از ابزار خودکاری که احتمالا فرد دیگری منتشر کرده برای عبور از اون و بازسازی مجدد فایل باینری اصلی استفاده میکنه . در این مسیر ابزارهای کمکی متعددی نیز وجود دارند که این روند رو تقریبا به یک وظیفهء روتین و نسبتا ساده مبدل میکنند ، فی الواقع برای UnPack اغلب راه حلهای آزاد یا تجاری موجود ، به چیزی حدود [حداکثر] نیم ساعت وقت نیاز است!

اما با این تفصیل ، استفاده از Packer همچنان یک گزینهء مثبت است . چون اگر انتشار باینری به همان شکلی که هست بدتر باشه ، انتشار نسخهء Packed اون با [حتی] یک Packer تابلو و غیر حرفه ای بد است و آدم عاقل میدونه بین این دو کدوم رو انتخاب کنه ؛ معهذنا همیشه یک حالت بهتر وجود داره .

برای حفاظت از امنیت یک نرم افزار به کمک Packer ها باید به چند نکته مقدماتی توجه کنید:

- ❖ Packer اولین سنگر حفاظتی است ، نه جدی ترین ، نه آخرین .
- ❖ Packer برای ایجاد وقفه در مسیر سهل تخریب امنیت نرم افزار مورد استفاده قرار میگیره نه چیز دیگر .
- ❖ توسعهء یک Packer شخصی و قابل قبول کار ساده ای است اما نه برای عامه توسعه گران
- ❖ حتی با استفاده از یک Packer خصوصی ، به امنیت غیر قابل دسترسی نخواهیم رسید

روش غلط استفاده از Packer:

- برنامه ات رو بنویس .
- یک Packer تابلو از یک سایت روسی یا چینی دودر کن .
- برنامه ات رو Pack کن و منتشرش کن .
- به دوست دخترت زنگ بزن و شب باهش قرار بگذار .
- نیم ساعت قبل از قرار به نزدیکترین ATM مراجعه کن تا پولی که از برنامه ات درآوردی بگیری و امشب خرجش کنی .
- بعد از مشاهدهء حساب خالی ات ، سریعا به فکر یک دروغ خفن و ی بهونهء ردیف برای دودر کردن عیال مربوطه باش

روش صحیح استفاده از Packer :

- برنامه ات رو بنویس .
- ضمن توسعهء برنامه ات با مطالعهء مداوم بخش امنیت نرم افزار سایت سعی کن کمی با ادبیات این حوزه آشنا بشی .



- با استفاده از اطلاعات منتشر شده ، دنبال یک Packer خوب و نسبتاً گرون قیمت بگرد
  - از یک نفر بخواه نسخه حرفه ای اون رو برات بخره ، به اسم خودت ثبتش کن و بلافاصله یک ایمیل برای توسعه گرش ارسال و ازش بخاطر نرم افزار خوش تشکر کن .
  - چند ساعت بعد تو یک ایمیل دیگه در مورد مشخصات برنامه ات براش توضیح بده و ازش بخواه بهترین توصیه هاش رو برات بفرسته و در پایان یک Best Wishes هم بنویس .
  - با استفاده از توصیه های او برنامه ات رو Pack کن .
  - با استفاده از یک HEX Editor قسمتهای به خصوصی از باینری ات رو تغییر بده . این کار باید باعث نا شناخته موندن Packer مورد استفاده بشه ؛ با مطالعه مقاله های موجود و مطالبی که فرومهایی مثل اینجا ارائه میکنند تمام تلاشت رو برای تغییر شکل و قالب نرم افزار بکار ببند و در این مسیر از هیچ ابزار عمومی و خودکاری استفاده نکن ؛ اگر نمیدونی چطور باید اینکار رو بکنی منتظر مقاله هائی که به همین منظور همینجا منتشر خواهند شد بمون .
  - برنامه ات رو منتشر کن ؛ به دوست دخترت زنگ بزن و شب باهاش قرار بگذار .
  - نیم ساعت قبل از قرار به نزدیکترین ATM مراجعه کن تا پولی که از فروش برنامه در آوردی رو خرج کنی
- درسته پولی که تو حسابته کمه ، اما احتمالاً اون شب بخیر خواهد گذشت . شاید برای برنامه های بعدی ات وقت بیشتری بگذاری ، مطالب بیشتری بخونی و از افراد با سواد تری کمک بگیری ؛
- بالاخره آدم به امید زنده است .

Packer ها عموماً تأثیرات به خصوصی هم روی نرم افزارها می‌گذارند که لازمه در مورد بعضی از

این تأثیرات فکر کنید:

- سرعت Packer : ها به شکل چشمگیری سرعت اجرا و سرویس دهی برنامه رو کم میکنند ؛ اگر داری یک سرویس مینویسی ، یا درایور سطح کرنلی داری که قراره با یک کارت PCI مرتبط باشه ، استفاده از Packer قابل توصیه نیست.
- اعتماد : در بسیاری از حوزه ها ، اعتماد به ماهیت نرم افزار ، نقش مهمی در توسعه اون ایفا میکنه . عموماً استفاده از Packer ضمن اینکه یک روش امنیتی محسوب میشه ، روشی برای مخفی سازی عملکرد نرم افزار نیز به شمار میاد.

Packer های خوب و سورس آزادی مانند yP و Packer های تجاری و قدرتمندی مثل SVKP همیشه در دسترس هستند و درگیری های خونین و ناموسی بین توسعه گران روشهای امنیتی و نفوذگران روی سایتهای زیر زمینی و فرومهای کذائی وجود داشته/داره/خواهد داشت ؛ اما برای یک توسعه گر آنچه نقش مهم و حیاتی رو ایفا میکنه ، کسب دانش و توانائی درک نحوه عملکرد ابزارهای امنیتی ، و اختصاصی سازی برخی از اونها برای دشوارتر کردن مسیر خدشه دار شدن حفاظ های نرم افزاری است . برای عمومیت برنامه نویسان ، کسب توانائی و تخصص فوق العاده بالا در امنیت نرم افزار نسبتاً امکان پذیر نیست ، چون کسب توانائی در این حوزه منوط به تعطیل شدن فعالیت در سایر حوزه هاست ؛ اما مطالعه مقدمات و آشنائی با ادبیات و کسب توانائیهای ابتدائی ، یقیناً یک وظیفه غیر قابل چشم پوشی است.

## Encryptor

- Encryptor ها جدی ترین چالش حال حاضر حوزه امنیت نرم افزار هستند .

Encryptor نرم افزاری است که عمدتاً و عموماً:

با رمزنگاری نسخه باینری روی دیسک فایل‌های اجرایی امکان بررسی و مطالعه اون رو تا حد زیادی از بین می‌بیره

با رمزنگاری/رمزگشائی دائمی و تو در توی توابع برنامه - نسخه موجود روی حافظه - بررسی عملکرد برنامه رو دشوار می‌کنه

با تغییر شکل متغیرها-نام و ادرس توابع - ارجاعات و کتابخانه های متصل شده بصورت استاتیک آنالیز نرم افزار رو دشوار می‌کنه

**Encryptor** ها رو همیشه به سه دسته کلی تقسیم کرد:

دسته اول: راه حل هایی که از روشهای متقارن برای حفاظت از کد استفاده می کنند.

دسته دوم: راه حل هایی که از روشهای غیر متقارن برای حفاظت از کد استفاده می کنند.

دسته سوم: راه حل هایی که از ترکیب دو راه حل اول یا استفاده از راه حل های ابتکاری استفاده می کنند .

اغلب Encryptor های تجاری موجود روش اول رو حمایت میکنند ؛ به عنوان مثال به توسعه گر اجازه میدن از یک 3DES و یک گواهی دیجیتال برای رمزنگاری توابع یا کلاسها یا ماژولهای خاصی استفاده کند ، و معدودی از اونها امکان استفاده از روش دوم یعنی اتکاء بر PKI و روشهای غیر متقارن رو نیز به این حمایت اضافه میکنند . به این ترتیب فقط ماژولهای مجاز میتونن ماژولهای مورد نظر رو از حالت رمز شده خارج و به حافظه فراخوانی کنند یا روی حافظه اونها رو وارد روتینهای رمزگشائی کنند ؛ و روند تصدیق هویت ماژول ( آیا این واقعا خود نرم افزار است که میخواهد بخشی از آن از حالت رمز خارج شود ؟ یا فرد دیگری ؟ ) و احراز یکپارچگی ( آیا

بخشی از کد یا حافظه تغییر داده شده ؟ ) عطف به مولفه های PKI بر عهده چهار عنصر CA: و گواهی دیجیتال و امضای دیجیتال و روشهای رمزنگاری است.

مثال:

برنامه A از سه ماژول a و b و c تشکیل شده . بخش اعتبار سنجی برنامه در ماژول C و برخی از توابع پایه ای نرم افزار که در تمام نرم افزار از آنها استفاده شده در ماژول a پیاده سازی شده اند . برنامه باید فقط به کاربر مجازی که فایل لیسانس معتبر رو خریده اجازه کاربری بده .

### سناریوی متقارن:

ماژول C با یک روش متقارن بصورت رمز شده منتشر میشه . توابع رمزنگاری بصورت توزیع شده در فایل اصلی برنامه و ماژولهای a و b پخش میشن ؛ بین هر دو تابع متصل نباید یک ارتباط طولی وجود داشته باشه ، یعنی نباید هر دو تابع متصل در یک ماژول باشند . برنامه اصلی موقع اعتبارسنجی توابع لازم رو بصورت دینامیک وارد حافظه میکنه ، با بررسی صحت تابع فراخوانی کننده و ماژول محاط اون ، ماژول b رمزگشائی و فایل لیسانس رو بررسی میکنه .

### سناریوی نامتقارن:

ماژول d یک CA است ؛ ماژول e هم حاوی هر دو الگوریتم مورد استفاده . برنامه اصلی با ارائه گواهی دیجیتال به d مجوز فراخوانی C به حافظه رو کسب میکنه و بعد از فراخوانی با بررسی گواهی دیجیتال C در چند نوبت از یکپارچگی اون مطمئن میشه ؛ توابع C پس از هر بار فراخوانی با بررسی گواهی دیجیتال برنامه اصلی صحت هویت فراخوان رو بررسی می کنند .

### فوائد روشهای متقارن:

-سهولت در پیاده سازی

-سرعت بالا در اجرا و ارائه خدمت

-کمیت قابل قبول بار اضافی تحمیل شده به برنامه

### نواقص روشهای متقارن:

- اطمینان دو طرفه وجود ندارد (تمام اتکاء روی الگوریتم است ، نه منطق اعتماد سازی مانند PKI شناسائی الگوریتم چنان که مینماید دشوار نیست و دور زدن اون نیز هم ؛ خلاء روشهای متقارن ، عرفی ترین عنصر این حوزه ، یعنی روند صحیح اعتماد سازی است).

### فوائد روشهای نامتقارن:

منطق قدرتمند و قابل اعتماد

### نواقص روشهای نا متقارن:

- دشواریهای پیاده سازی و کاربرد
- هزینه
- کمیت بالای فشار بار اضافی تحمیل شده بر نرم افزار
- افزایش پیچیدگی اجرا : کاهش چشمگیر سرعت ارائه خدمت

Encryptor ها هم مانند بقیه مولفه های نرم افزاری ، میتونن در صورت کاربرد صحیح و معقول ، تا حد قابل قبولی ضریب اطمینان نرم افزار رو بالا ببرند و - باز هم - مانند سایر راه حلها ، هیچوقت هیچگونه تضمین عینی و قطعی برای حفاظت صد در صد از هیچ چیز ارائه نمی کنند.

در ادامه ، ضمن معرفی چند کمپرسور Packer - و Encryptor و شرح مختصر روش کار هر کدام ، بیشتر در مورد ویژگیهای یک راه حل قابل اعتماد حرف میزنیم ؛ احتمالاً آخرین مطلب درس دوم هم به بررسی جزئی دو سناریوی کامل استفاده از این راه حلها باشه.

لیست اجمالی از کمپرسور/پکرهای متداول که بصورت عمومی در دسترس هستند ؛ برخی تجاری و برخی رایگان ؛ برای کسب اطلاعات بیشتر سایت هر کدام را ببینید:

<http://www.anticracking.sk/download.html>

SEPP

<http://www.webtoolmaster.com/exes.htm>

EXE Stealth

[http://www.anticracking.sk/products\\_svkp.html](http://www.anticracking.sk/products_svkp.html)

SVK Protector

<http://www.rtsoftware.org>

Code-Lock

<http://pelock.pac.pl>

PE Lock

<http://www.xprotector.com/downloads.php>

Themida

<http://www.aspack.com>

ASProtect

<http://www.siliconrealms.com/armadillo.shtml>

Armadillo

[http://www.pc-guard.co.yu/e\\_pcgw32.htm](http://www.pc-guard.co.yu/e_pcgw32.htm)

PC Guard

<http://www.ultraprotect.com/download.htm>

Ultra Protect

<http://www.obsidium.de/show.php?details>

obsidium

<http://virogen.cjb.net>

VGCrypt

<http://yodap.cjb.net>

Yoda

<http://egoiste.da.ru>

tElock

<http://pespin.w.interia.pl>

PESpin

<http://www.blinkinc.com>

Shrinker

<http://www.neoworx.com>

NeoLite

<http://www.collakesoftware.com>

PEBundle

<http://www.pecompact.com>

PEcompact

<http://packman.cjb.net>

PackMan

<http://upx.sourceforge.net>

UPX

<http://dwing.go.nease.net>

UPack

بخاطر داشته باشید که برای تمام این ابزارها و ابزارهایی که به مرور به این لیست اضافه میشوند راه  
حلهای خودکار عبور و لغو کارائی های امنیتی وجود دارد.



## ParsBook.Org

پارس بوک، بزرگترین کتابخانه الکترونیکی فارسی زبان

# ParsBook.Org



The Best Persian Book Library